Vom Fachbereich für Mathematik und Informatik
der Technischen Universität Braunschweig
genehmigte Dissertation
zur Erlangung des Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von
Dipl.-Inform. Dirk Sturzebecher

A portable and flexible Framework for CSCW Systems

Datum: 01.11.2000
Referenten: Prof. Dr. M. Zitterbart, TU Braunschweig, Deutschland
Prof. Ph.D. S. Hariri, University of Arizona Tucson, USA

Preface

Sitting here and thinking about an appropriate preface for this document makes me think about chances and the importance of some choices, especially those which at the time seamed to be of little consequence. In this moment the one choice coming directly to my mind was the selection of a seminar about High Performance Communication. The reason to chose this specific seminar was purely its date. It seamed to be a rather unimportant choice about five or six years ago. Despite some problems on the way this lead to accepting a Diplomarbeit at the same institute some time later. And again this choice was not just dictated purely by the topic or the by then acquired (good) opinion of this specific institute, but as well by a number of other conditions. At that stage I would not have thought to do (further) postgrad studies. I actually guess nobody did expect this, not me, not my friends and not my parents. Again live turned out to be different than expected, as many times before and I signed up as a postgrad doing my doctorate at the Institute of Operating Systems and Computer Networks. Thus, as a result, some timetable considerations did change the rest of my life! Goes to show that after all the little things are those which end up making live interesting.

Over the last three and a half years I had the opportunity to work in the very interesting and fast developing field of computer supported collaborative work. The MACS project allowed me to realize many of my ideas, test others and refine them. Now having achieved to build quite a nice system it is in a way rather sad to move on to something new, as so many interesting topics remain open for further investigation. On the other hand it is as well time to move on, as I have been getting restless again.

These last years have been rather interesting, not only on the academic side or the development of MACS itself, but as well due to a great number of other influences. The opportunity of working in a European Project thought me many new skills which would have otherwise not been accessible to me. This was not always easy. At times I had the feeling that I should change my degree to cover economics or public relation instead of computer science, but in the end it was worth it, especially considering the insights gained.

At this point I really wish to thank Prof. Dr. M. Zitterbart for providing me with the opportunity to join her group, work on my doctorate in the frame of the MACS project, the arrangement of the European Project (despite it being totally different from the expected) and for the outstanding support received during the whole time!

Further I want to thank the whole MACS team for the support and work put into the project. Without the many students and my colleges working in this field MACS would not have been as successful.

Fortunately I did not only have support on the academic side, but as well by my friends on a personal level. A special thanks goes to those who have “suffered” with me and shared my “fate”. And last, but not least, I like to thank my parents who made all of this possible by their support.

Abstract

The rapid development of computer networks, the tremendous acceptance of the Internet and the increasing pace of change of communication itself have strongly influenced our lives over the last few years. New ways to communicate and new structures for organizations and companies have emerged. “Virtual” is one of the buss words of our time. In order to incorporate all these changes and as well to help in their development new applications supporting these structures are needed.

In this work a framework designed to provide the flexibility, portability and scalability to support these new applications is introduced. Its flexibility allows easy and efficient development of modules and applications for specific usage scenarios, while at the same time providing implementations of basic functionality, as well as, some more advanced examples. A usage scenario specific system can thus easily be created by a combination of specifically developed modules and standard application modules provided by the framework. The approach taken hereby is to combine as much common functionality as possible in the framework itself and hence to reduce the complexity for the developer. A number of different scenarios are directly supported by the framework, while others can easily be added. These scenarios cover a wide range of situations and therefore demonstrate the flexibility and scalability of the framework itself.

This system is called MACS (Modular Advanced Collaboration System) and a combined effort of the MACS team. The parts of MACS treated in this document refer mainly to the framework design, placing special importance on the control mechanisms, modularization and scalability, as well as on interworking capabilities with other well known systems.

Abstrakt

Die rasante Entwicklung von Computernetzwerken, die unglaubliche Akzeptanz des Internets und die stetig steigende Geschwindigkeit mit der sich unsere moderne Kommunikationswelt ändert beeinflußt längst weitreichend unser Leben. Neue Wege der Kommunikation und der Organisation, vor allem auch von Firmen, haben sich entwickelt. „Virtuell“ ist eins der Modewörter unserer Zeit. Um allen diesen Veränderungen Rechnung zu tragen, ja sogar mitzuhelfen diese zu gestalten, sind neue Anwendungen zur Kommunikation nötig.

In dieser Arbeit wird das Design von einem Rahmenwerk vorgestellt, dass speziell für solche neuen Anwendungen entwickelt wurde. Es zeichnet sich besonders durch seine Flexibilität, Portabilität und Skalierbarkeit aus und ermöglicht somit eine einfache und effiziente Anpassung an spezifische Einsatzgebiete. Dieses wird durch die Integration entsprechender für das Einsatzgebiet spezialisierter Module erreicht. Durch die Zusammenstellung von vorhandene Modulen mit Basisfunktionlität und weiteren themenspezifischen Modulen läßt sich ein auf das jeweilige Einsatzgebiet zugeschnittenes System erstellen. Hierbei wird möglichst viel der modulübergreifenden Funktionalität durch das Rahmenwerk bereitgestellt, und somit die Komplexität und der Aufwand für die Entwicklung neuer Module drastisch reduziert. Weiter wird eine Reihe von Einsatzszenarien direkt durch vorhandene Module unterstützt, während Anpassungen für weitere Szenarien leicht möglich sind. Dieses ist ein gutes Beispiel für die mit diesem Rahmenwerk erreichten Flexibilität und Skalierbarkeit.

Das hier beschriebene System nennt sich MACS (Modular Advanced Collaboration System). Im Rahmen dieser Arbeit wird vor allem auf das Design des Rahmenwerk und der Anwendungs- und Systemerweiterungsmodule eingegangen, sowie auf die verschiedenen integrierten Kontrollmechanismen und Dienste. Auch wird die in MACS erreichte Portabilität, Flexibilität und Skalierbarkeit, sowie die vorhandenen Interworking Fähigkeiten detailliert beschrieben.

Content

Figures

Listings

Tables

1 Introduction

1.1 Motivation

During the last few years the rapid development of computer networks and the tremendous acceptance and growth of the Internet have caused drastic changes to the way people interact. While some forms of interactions have matured to every day usage (e.g., email and WWW) others are still under development. An example is the support for geographically separated people working together across computer networks. The support for such group work is in existing systems fairly limited and inflexible. For such scenarios many relevant aspects are still research topics and not yet considered in commercially available systems.

With the change of the way people interact via computer networks their requirements regarding these are changing. New applications are emerging requiring support by specialized network services. Today's data network services which are mainly based on a simple best effort approach for transmitting data packets will have to evolve to a more sophisticated form. Future networks may not only possess data transmission capabilities, but as well include processing and storage functionality [71]. Some approaches to achieve this type of functionality are currently under development, such as, for example, flexible network infrastructures realized by Active Networks [25] [75]. These do add processing capabilities to (some) nodes in the network, allowing the modification of data passing through them. A simple example would be a node changing the quality of a video stream for a specific receiver. This would, for example, allow a PDA to receive a specifically modified video stream. Generally, due to the small screen size and the highly limited connection bandwidth a PDA will only be able to receive a video stream if it is drastically scaled down, has reduced colors and a reduced frame rate.

The added intelligence in the network can, thus, be used to provide a customer specific service. The fast, efficient and flexible creation of such network services is a first and important step towards offering a more general customer orientated communication support. Not only the network must evolve but as well the applications running on top of it. This new generation off applications will have to provide a high degree of flexibility to meet the fast changing user requirements and to take advantage of the resources offered by the network. As a result a number of building blocks typically found in today's applications will have to migrate into an application development framework or will actually have to be placed in the network itself. Only such an approach will allow the fast, efficient and flexible creation of customer specific applications, and therefore provide a infrastructure helping the developers to meet the rapidly changing and highly specific requirements of these new applications.

1.2 Scenario

To illustrate the environment for these new types of applications one can consider the following example scenario of a large automotive company developing a new product.

Large companies are distributed over many different countries with different sites in most of them (e.g., Figure 1). Even within a single plant different parts of the organization will have to cooperate in the development process. What is needed to support such a process?

There has to be some form of shared database (Figure 1, company head quarter) for documents, such as construction plans, part lists and even simple things as lists of appointments. Access to these is normally asynchronous, as they will be requested and

modified by different people at different times. To conduct working group meetings in order to plan, develop or evaluate a specific task, support for synchronous communication between development team members is necessary. Here the easiest approach is the telephone, but this will not suffice for larger groups. In large companies the use of video conferences is already well established. These are normally conducted in special rooms with dedicated hardware (large companies, such as, for example, Volkswagen have specific conference centers linked up via satellite). It is not yet common to conduct video conferences right from the normal workstation or PC of the participants. While video conferencing is a step towards a faster coordination and information exchange it does not really allow the participants to work together simultaneously. Normally people will work on their own according to the decisions of the last meeting, but not together. To achieve a cooperative simultaneous working environment the tools used to work must be coupled to allow the simultaneous modification of data by different participants. The videoconference then only serves as a basic communication tool, but is not the actual center of attention. How would such a simultaneous development process look like?

Taking, for example, an engineer, a designer and a project manager, they could meet virtually to discuss a problem in the current product development. The engineer might want some changes in order to improve a certain aspect of the product. This might have both consequences on the cost (responsibility of project manager) and on the design (responsibility of the designer) due to a changed size of the part under consideration. Working together in a shared environment will allow them to see the implications of the modification made by any one of them immediately and thus allow the others to directly validate them (Figure 1, individual). As a result a solution will be achieved faster than by conventional methods. That is, conventionally one of the involved persons would modify the relevant aspects according to his expertise and then send the modifications to the others to study. They will make modifications again and send them back. Before the modifications are finally acceptable to all of them many iterations may be necessary.

While this illustrates well the advantages of collaborative work it does not yet highlight the advantages gained by a development framework for such applications or a flexible and intelligent network infrastructure. An important point is, that the different parts of the system have greatly varying requirements, which must be met in order for the whole system to function properly. For example, the transmission of the modified plans via email is uncritical, it makes little difference if the plans arrive some seconds earlier or later. On the other hand the transmission of audio and video poses hard constrains on transmission delay and therefore the network. The videoconferencing part of the synchronous work is only there to support the various applications and therefore is a typical component which should be easily reusable. Such general purpose components should be offered by the framework used to develop the actual application. The data processed by the actual application may again vary greatly, it might, for example, be based on some textual representation of a part list, or it might as well be part of a 3D virtual environment used during mockup. Latter application is time critical and requires a large amount of network resources. To meet these requirements currently dedicated lines are used, but with a resource reservation the normal Intra-/Internet structure would provide a much more cost effective solution.

Here development frameworks and underlying flexible networks can provide a more efficient environment for the development of collaborative applications by providing typical functionality in reusable components, as well as allowing the easy integration off application specific modules. A further advantage of such frameworks can be seen in versioning. Here basic development tasks are the same, only slight differences exist. This can range from something rather cosmetic, such as the corresponding project logo, to much more substantial

components which are integrated into a currently existing system (or do replace components of it). A good example of versioning are the Java editions provided by Sun. All three, the "Enterprise Edition", the "Standard Edition" and the "Micro Edition" use many of the same components and share in parts common interfaces, but each is aimed at a different target group and, thus, provides further functionality and tools relevant for this group. Another example better fitting the previously described scenario are the two station wagons variants of the Golf and Bora as build by Volkswagen. They are basically the same car. The Golf is mainly aimed at families. The Bora is on the other hand labeled as a "life-style" car and features better interior, a stronger default engine and includes many of the additional extras of the Golf version by default.

Figure 1 shows the basic setup of the above described automobile company scenario including the distribution of plants, the company head quarter with the database and network connections and the requirements of the individual workspace.

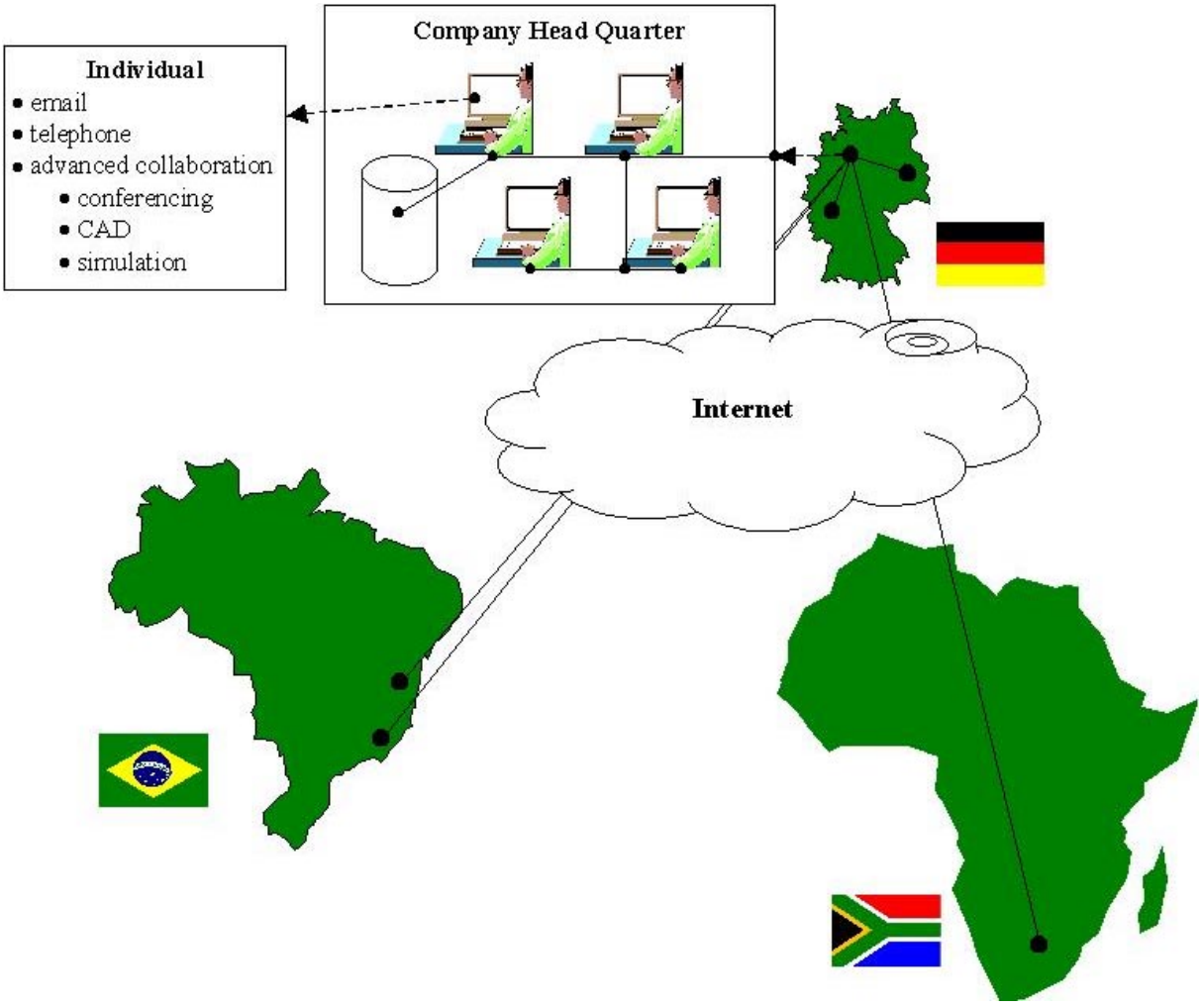


Figure 1: Example of inter-location collaboration in a large, international company

1.3 Aims of the MACS framework

To support a complex scenario as described above it is necessary to design a highly flexible, scaleable and portable framework, providing basic application modules and allowing an easy integration of usage specific applications. These would be in the above example specialized

tools supporting the automotive development process, covering areas from construction (e.g., CAD) to company specific management structures (e.g., supply chain). To integrate such a complex system into a company requires a lot of time and effort, raising the question of interworking with existing systems. This is not only relevant during the transition phase within the company, but as well relevant to allow the communication with entities outside the company, such as its suppliers.

The development of MACS (Modular Advanced Collaboration System) thus aims at providing the required flexibility, scalability and portability, while at the same time supporting interworking with existing and commonly used conferencing systems.

While current systems mainly offer flexibility with respect to their user interface (interface can be replaced by specific implementation) and by their supported media (mainly audio and video codecs), MACS flexibility extends through all offered functionality, including the framework itself, the available applications, the supported network protocols and even the graphical representation of the communication management. To achieve this MACS uses a highly modular approach. Modules have well defined interfaces allowing them to be fitted together according to current requirements. A clear structure dividing the system in various levels with well defined functionality allows easy modification of existing modules or the integration of new usage scenario specific ones.

Scalability is a very important issue in order to allow a CSCW system to be used in a complex deployment environment. It must cater for communication groups exhibiting great differences regarding the number of participants, as well as their geographical distribution. Group size alone may vary from a small discussion group with maybe 10 participants to large podium discussions with many hundreds or even thousands of participants. It is not possible to create only one set of modules covering the entire range, as there is no *one-size-fits-all* solution. This is not only true for the management of the participants of such a group itself, but as well applies for the application modules used. MACS therefore provides different modules tailored to meet the requirements of typical usage scenarios. For these usage scenarios it is not sufficient to only consider the scalability aspects of the framework itself, but further issues ranging from the network support to the actual graphical representation must be regarded.

Portability is a key aspect to support deployment in the heterogeneous computer environments typically encountered today. MACS relies on the Java language to achieve a high degree of portability. This was already during the development of MACS a necessity, as MACS was developed in a heterogeneous environment, using different computer hardware running different operating systems, such as Windows, Linux [47] and Solaris.

Interworking is supported through a set of modules, highlighting the practical advantages of the modular design of MACS and the thus achieved flexibility. These modules are optional and are only activated on demand.

Considering the above introduced aspects of MACS the overall complexity of this project clearly exceeds the capabilities (especially in man power) of any single person. MACS is a joined effort of a whole team. My contribution to MACS, as described in this document, is based on the design of the overall framework, especially considering flexibility and scalability. Key aspects of this are the modularization and configuration according to user requirements and usage scenario for all components within the framework, especially the main control unit and the transaction system. Further included are some of the applications, mainly whiteboard and feedback, and to a lesser extend audio and video, as well as the internal modules for interworking with other CSCW systems. Visualization and network aspects are not fully covered, but referred to in parts. The investigation of management of

groups is limited to the mechanisms needed, but does not include any management strategies or group visualization issues. These issues are considered in a different thesis doctoral [19].

1.4 Structure of document

Chapter 2 describes the fundamental issues of CSCW systems and introduces a reduction in scope of the CSCW systems relevant for this work. Furthermore control issues within a CSCW system are introduced. A comparison of current CSCW systems representing the state of the art concludes chapter 2. Chapter 3 introduces the concepts and design employed for the MACS system, discussing important aspects starting with the structure of MACS and covering topics such as scalability and modularity. Chapter 4 covers basic support applications provided with the MACS framework. Chapter 5 discusses interworking. Chapter 6 highlights selected key aspects of the implementation including a performance analysis and some real world usage experience. Chapter 7 contains the conclusions and ends in an outlook on possible further developments.

2 Fundamental issues of CSCW systems

This chapter provides a structured overview of the relevant aspects for CSCW systems. First, CSCW systems in general are introduced and some typical usage scenarios are discussed. These are then used to analyze control aspects and to consider scalability issues. The chapter concludes with an overview of the state of the art systems, highlighting some typical limitations.

2.1 CSCW Systems

CSCW stands for Computer Supported Cooperative Work. This term covers any computer supported system allowing people to work together hopefully in a cooperative manner and aimed at achieving a common goal. Such systems can range from something as simple as email based systems (chapter 7.2 in [97] and [84]) to highly customized and complex systems specialized for a very specific task. Typically a CSCW system integrates different aspects [41] and hence covers a wide range of activities as, for example, described by the introductory scenario. A very important point to note is, that the CSCW field is interdisciplinary in nature [51]. Therefore these systems can support cooperative work, but they can not enforce it [11]. The will of people to cooperate is not dependent on the system in use. The question whether people will cooperate is not in the scope of computer science and, thus, not treated in this document. Only the computer science relevant aspects of the provision for a CSCW systems will further be regarded. Hence the question treated here is how to support people working together towards a common goal in an efficient and cooperative manner, and not how to motivate them to do so.

In order to limit the scope of this work different types of CSCW systems are defined. The investigation is then focused on a specific type of system analyzing their functionality and components. Further typical tools of such environments are discussed as well as examples of more specific ones.

2.1.1 Types of CSCW Systems

As CSCW systems cover a wide range of applications [11] [22] it is useful to define some types of systems and, thus, limit further discussion to a specific subset. A straight forward division is by the operation mode of a CSCW system:

- synchronous
- asynchronous

In synchronous systems users are working together at the same time, interacting with a potential high frequency. They are generally able to directly observe the modifications done by other users and to react to them immediately. In an asynchronous system users might still work at the same time, but this is not a requirement and they may even not be aware of it. Modifications done by a user are thus not immediately visible to others. As a result two different users might perform conflicting modifications independently of each other on the same document. This can only happen if no locking mechanism exists to prevent users from working on the same document or object at the same time. In such a case the system can not guarantee data consistency. As a result during the next synchronization conflicts might occur. A locking mechanism will mark the section or file worked on as being edited. Other users

wanting to edit the same section or file will be warned of possible conflicts, or the system might even restrict access to this section/file to read only mode. This ensures that no conflicting changes can be performed and hence data consistency is guaranteed, as no two or more different versions of the same section/file ever exist.

Further systems can be

- dedicated,
- general purpose or
- customizable.

Dedicated systems serve only a specific aim or work step. They are specifically designed for that task and are as well limited to it. An example are systems used to process configurations, as can be found in ordering systems. Here the configuration of one very specific product can be modified, but the system is limited to exactly this product. Taking the automotive scenario again as an example the system used to specify the configuration of a specific car (e.g., a VW Bora) for ordering is dedicated to this task. It might actually be dedicated to work only for this car, but may be able to handle similar cars as well (e.g., VW Golf, as it is the same platform), but certainly will not be able to handle cars in general (i.e., cars from other manufacturers) or even the complete product list of a single manufacturer including all the accessories (e.g., tires, radios and up to T-shirts with the company logo).

General purpose systems on the other hand do not possess a detailed knowledge of the product/issue under consideration. An example is a video conferencing system which only provides communication support, but make no assumption about the content communicated and thus can be used to support any given task. Or taking the above example of a ordering system, a general purpose system would allow you to order a wide variety of products, such as cars from many manufacturers, as well as other products.

Customizable systems are general purpose systems, which can be modified to better support a specific task. In an extreme case a customizable system can start out as a general purpose system and be modified to become a dedicated system. The example would be to start with a general order system able to handle any product. This system is then customized to take care of the special requirements of, for example, a car ordering system. This is only a customization, but if the system is further specialized to a specific brand, even more to exactly one specific model, then it becomes a dedicated system. Table 1 shows in how far systems of these categories are relevant to the CSCW framework presented in this document.

	dedicated	**general**	**customizable**
asynchronous	not relevant	not relevant	not relevant
synchronous	not relevant	relevant	relevant

Table 1: Types of CSCW systems considered in MACS

For this document mainly the synchronous not dedicated CSCW systems running on a normal multimedia PC or workstation are of relevance. Therefore, from here on only this aspect of

CSCW systems will be investigated and references will, if not specifically stated otherwise, always refer to such systems.

2.1.2 Basic characteristics and functionality of CSCW systems

CSCW systems under consideration in this work share certain common components and functionality.

For a user to use the system it is necessary to run a copy of the system locally on the users own PC or workstation. This is called an **instance**. Normally the instance is tied to the specific user who started the system.

To collaborate it is necessary to select a number of other users to form a group. The frame for the communication of this group is called a **session**. A session is created by a single user. Others will then be invited into the session or join the session by themselves. A user can be in more than one sessions at any given time. Sessions can have different characteristics. Typical ones are:

- open vs. closed
- moderated vs. not moderated
- encrypted vs. unencrypted

Open sessions have no access restrictions, while closed sessions only allow certain persons to join. The access restriction can either be explicit or implicit. Explicit generally refers to passing join request, that is a request to be allowed to join the session, on to the **session chair.** The session chair is the person in charge of the session. He will then decide whether to allow the requesting person to join the session. In the implicit case the session chair is not involved, but the system uses a set of rules to reach a decision regarding the join request. Such rules, for example, might request a password or limit access to certain regions/domains. The later is useful, for example, to limit access of information to members of a certain group within a company or as defined by a geographical region. This can become necessary, for example, due to legal considerations not allowing residence from certain countries to access specific data.

Further a session may be moderated. In this case someone, normally the session chair, has the control over the session. This allows him to set access restrictions, to expel participants and to control the floor policies. This will be the case for any session requiring structuring through a leader (the session chair), as is typical the case in class room/lecturing scenarios.

Encryption is especially important if confidential topics are discussed within a session, as sessions conducted via the Internet or any other public network can relatively easily be monitored. This is nearly always relevant for companies, especially considering the increasing efforts made to integrate suppliers via networks further into the company structures. A good example for this is again the automotive industry, as here suppliers are highly integrated. In newer factories they are even responsible for the end assembly of their specific parts into the product itself.

The session forms the frame for the use of the locally installed **application modules**. Each of these modules allows a certain type of work to be performed. Users must have the same module or at least a compatible one installed in order to participate in the specific work. Examples for such application modules are:

- a drawing module to share drawings (whiteboard)
- a text module to share text (chat)
- a media module to send and receive audio and video (audio and video)

In order to control the communication sequence, e.g., select a speaker for the audio transmission and prevent others from interrupting him, a schema enforcing rights/permissions to the audio channel is needed. Such rights/permissions are represented by an abstract object called **floor**. Each resource can have its own floor assigned to it and therefore have separately controllable rights and permissions. Each floor is unique to the session it is used in. A **resource** can be any real (e.g., a video camera) or abstract (e.g., a slide in a drawing application module) object specified within a session.

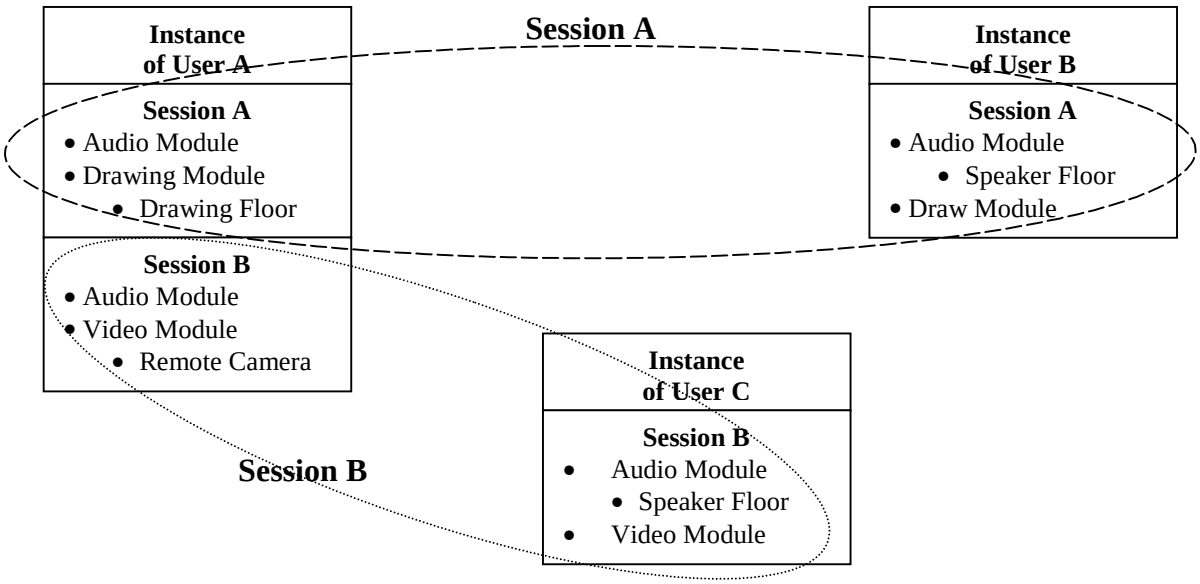


Figure 2: CSCW system components and functionality

Figure 2 shows a simple example of three users. User A participates in two sessions namely session A and session B each with different applications. Each of the sessions has one further user involved being user B and user C. Additionally a sample distribution of floors/resources of some of the application modules is shown. Here each floor/resource is unique within the corresponding session, that is no two users in the same session do posses the same floor/resource simultaneously. For example, the "Speaker Floor" of the audio module in session A is owned by user B, while for session B the "Speaker Floor" is owned by user C.

Taking up the automotive industry scenario this situation could, for example, arise if user A is discussing the integration of a part with a supplier represented by user B, but at the same time wants to check back concerning product cost implications. He does this by contacting the planing department represented by user C in a separate session. Setting up two different sessions is done to restrict the supplier from listening to internal potentially confidential information, as, for example, about costs and profits directly related to the part under consideration. If no restricted internal information is being discussed a single session with all three users participating would be more efficient.

2.1.3 Basic structure of CSCW systems

Which components are typically necessary to realize the above introduced functionality? By considering the following three functions offered by a CSCW system it is possible to identify some basic components. These functions are:

- session setup/tear-down
- session management
- application usage

Firstly, the session setup/tear-down relies on the user being able to select an existing session to join or to create a new session himself and to select a group of users to invite. This means there has to be a mechanism to propagate session and user information. This, as a minimum, requires a database to manage these information, some network functionality to actually propagate them and a user interface to display them.

Secondly, for the session management again at least a database to contain the session information is needed, as well as a user interface.

For the applications once more a user interface is required and some network functionality to exchange information related to application status.

Hence the following component are needed:

- network services
- control and databases containing the currently known sessions/users
- a user interface

It seams to be useful to consider the functionality of the applications separately, allowing different applications to work within the CSCW system. While each CSCW system will have these components, they are not necessarily clearly separated by the design of the system. Having only considered the most basic functionality it will, in general, be possible to identify further components.

It is as well useful to differentiate between a CSCW system and a CSCW **framework**. The framework offers a frame for the integration of applications into a CSCW system, by providing a number of services and functions allowing an easier and more efficient CSCW system development. The specific applications together with typical general purpose applications provided with the framework itself then form the actual CSCW system.

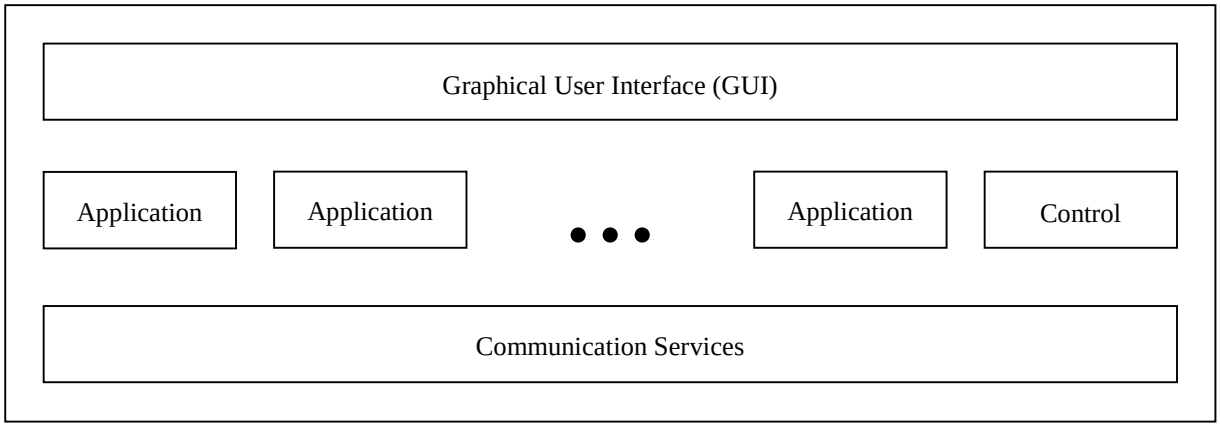


Figure 3: Typical structure of a CSCW systems

Thus, as shown in Figure 3, a CSCW system typically contains components realizing communication (communication services), implementing application functionality (application modules), providing control information and mechanisms (control), as well as components responsible for visualizing system information and control elements (GUI).

Taking a bit more detailed look at these, the following can be determined. The communication services provide access to the network and, in general, support a set of distinct protocols and services. These will have to fulfill a wide range of requirements extending from the simple initial example of propagating session and user information to the delivery of large amounts of real time data such as contained in a video stream. Further these are dependent on the underlying network technology and have to hide its details as far as possible from the higher levels (which can be achieved by an adequate object oriented approach [3]).

The application modules contain the main logic to implement the desired functionality of the corresponding **tool**. A tool should be mainly seen by its suitability to reach a well defined goal. Thus, a tool can be a specific component of the CSCW system or the actual CSCW system itself. The main point is that it is used to achieve a very specific goal. This is different from an **application module**, as the application module is normally designed to deal with a specific content and hence is more limited in scope as a tool. Only if the goal is to work on a specific content according to the functionality offered by an application module it itself becomes a tool. In a CSCW framework the application modules may rely on the same set of communication services as the rest of the CSCW system, but may as well provide their own application specific ones. An example would be the video application module. It will use basic networking services (e.g., UDP) provided by the communication services, but may layer on top of this some other protocol in order to better deal with its real time requirements (e.g., RTP), which may be specific to this application module and, thus, not found in the general services as provided by the communication services. Optionally the application modules may use the databases offered by the control to access user information. The content of these databases will vary greatly based on the system under consideration. Here a **closely coupled** system, that is a system tightly integrating all components and generally offering detailed control over these, will need more specific information about the applications and user, than a **loosely coupled** system, where there is little interconnection of the components and where the application modules may actually be completely independent of each other.

The GUI provides the user interface. It implements the visualization of the known sessions, users and applications, as well as the provision of control elements for the user to interact with.

2.1.4 Types of application modules

Each application module must implement its core logic itself. In general the application module should relay as far as possible on the services offered by the CSCW framework. This will allow a faster, more efficient and more reliable implementation due to the reuse of existing and well tested code. CSCW frameworks do typically provide some basic application modules which are not specific to any special usage scenario [43] as, for example, audio, video and whiteboard application modules. Sometimes a voting or feedback application module is provided as well, allowing to generate detailed feedback about an ongoing session or to vote on a current topic.

Specialized application mostly depend on the usage scenario. In a software development company CASE (Computer Aided Software Engineering) tools will be of relevance, while, for example, in the introductory scenario situated in the automotive sector CAD (Computer Aided Design) and specific simulation tools will be more relevant.

2.1.4.1 General application modules

Audio and video

In the context of MACS audio and video refers to interactive audio and video, thus, excluding considerations typically relevant for streaming music or movies.

Audio tools can have two basic flavors, the one-at-a-time and the many-simultaneous variants.

The first will only allow a single person to speak at any given time. This is not only the minimal usage of bandwidth, but for certain scenarios the mode of choice. In a formal presentation or a press conference it is rather a wanted feature that only one person speaks at any given time, others can "raise their hands" in order to be allowed to speak next.

The many-simultaneous variant is more normal for small meetings, were the behavior is mainly controlled by social protocols and must, therefore, not be enforced by the system itself. Here the bandwidth requirements are higher (each listener can simultaneously be a sender). To playback the received audio streams it is necessary to mix them, in order to hear the resulting conversation, or in the case of too many people talking at the same time the resulting noise.

Video tools normally exhibit two similar flavors, but the bandwidth requirements are even more severe. Due to this most of the times the one-at-a-time mode is used, limiting the video to the person speaking. Tools supporting meetings or lectures with many video streams (many-simultaneous) require more resources as today are commonly available.

Audio and Video tools are the basis for communication between the session participants in a CSCW system, but they in themselves only extend the capabilities of (video) telephony to cope with more participants at the same time and do not provide any additional assistance for discussing or working on any specific content.

From here on the combination of audio and video tools will be referred to as **media tools**.

Whiteboard

A whiteboard replaces real world mediums like a blackboard, flipchart, overhead- or slide-projector and laser pointer with a single tool suited for network based interactive collaboration [33]. Different graphic formats can be imported into a whiteboard as, for example, postscript, JPEG and GIF. Using the whiteboard one can create graphical primitives like lines, arrows, arcs, rectangles and text strings. This enables the participants of a session to prepare slides for a presentation with a tool of their choice, load them into the whiteboard and eventually enhance the slides during the presentation. It is possible to place a tele-marker on the drawing panel to focus the attention on a topics. The tele-marker is often implemented by a hand symbol which can be placed next to the text, just like placing a pen on an overhead slide. The user may scroll through the available pages and change their content or create new ones. A tele-pointer function enables the participants of a session to follow the mouse movements of the currently active user. Figure 4 shows a whiteboard containing a slide of a presentation with annotations and tele-marker.

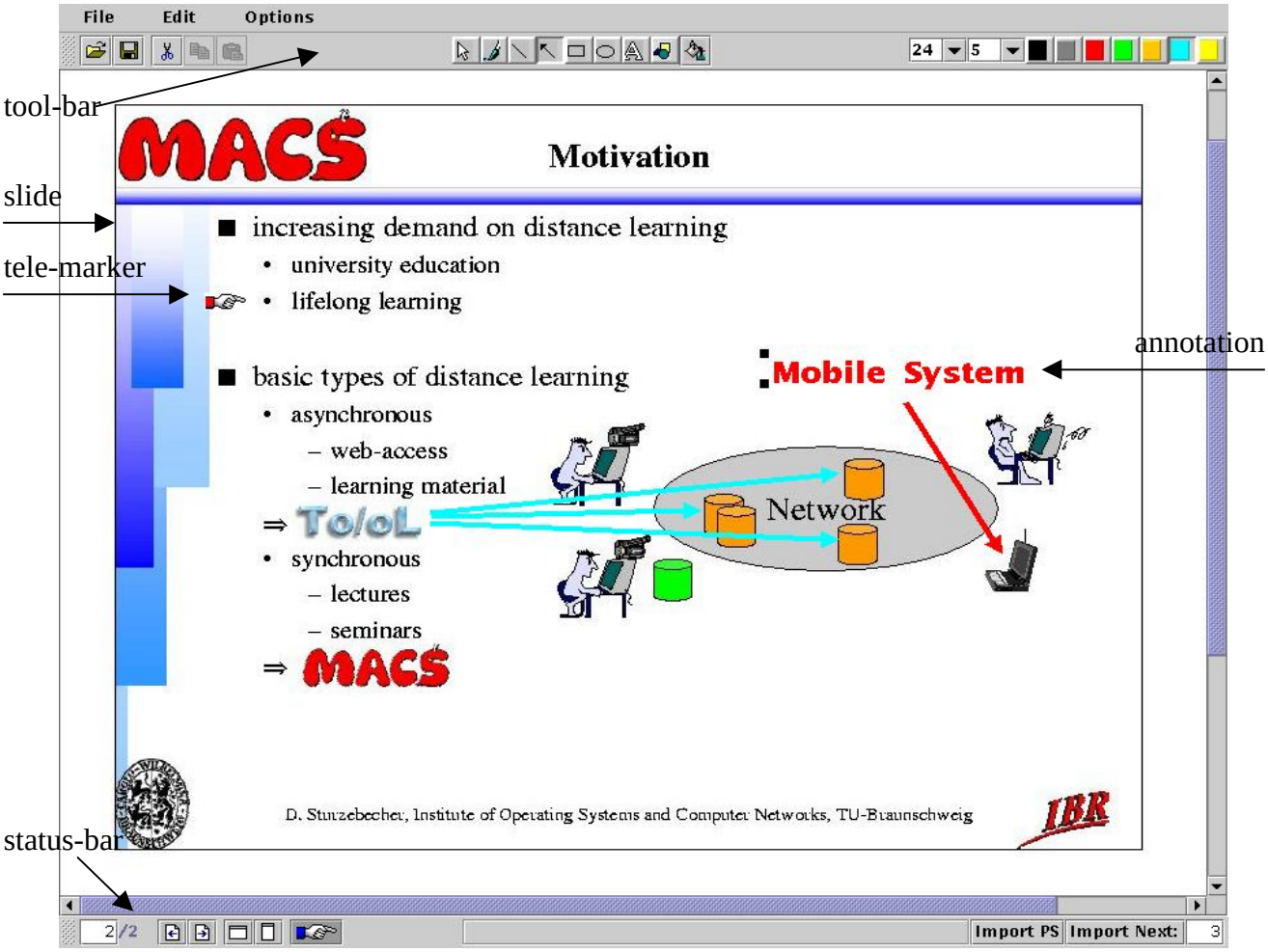


Figure 4: A whiteboard used for a presentation

The whiteboard actually extends the possibilities offered by CSCW system in comparison to telephony, as it provides a place to present slides during a meeting and allows to discuss these, including the ability of placing annotations and sketches on the presented slides. Still this is not scenario or content specific, as it can be used in the same way during a group meeting independent of whether the meeting discusses software development, automotive

design or planing problems in the production process. Whiteboards are as well typically used in learning scenarios [9].

Voting and feedback

Besides the explicit interaction via audio and video there are some mechanisms required to provide a somewhat more indirect feedback. Examples are voting or feedback tools. These do provide a common ground to rate something on. If, for example, a question is put up for voting during a lecture, the participants can enter their acceptance (yes) or declination (no) into the feedback tool. It will calculate the percentages and show them to the questioner. This does not have to be limited to simple yes/no questions. A useful approach is as well to ask for feedback on a running lecture. Here a rating can be given, for example, for audio/video quality, presentation speed and/or clarity. The lecturer has now a mechanism to get a feedback on how well the lecture is going. In traditional scenarios (i.e., in a real lecture room) the lecturer could extract (with some practice) these information from the participation and facial expression of the students. This possibility has been greatly reduced, as the audio and video links in such scenarios are of a comparable bad quality, if available at all (i.e., not all students will be able to send a video signal due to lack of equipment or bandwidth). See Figure 5 for an example of a feedback tool.

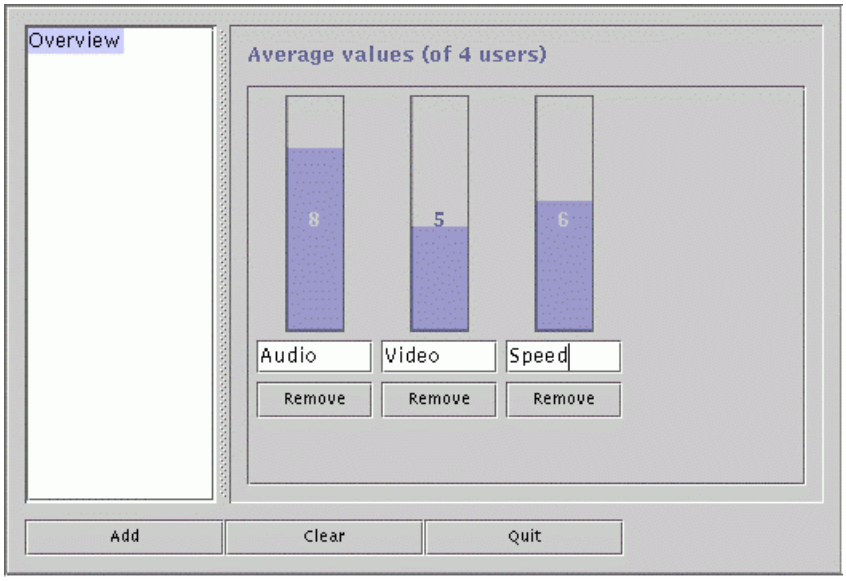


Figure 5: Feedback tool showing audio and video ratings

2.1.4.2 Specific application modules

Software engineering

Software engineering supported by CSCW systems offers a number of advantageous [23]. Teams can be formed by the experts best suited for the task at hand without considering their location or the travel overhead. As software engineers are computer experts they should be able to handle even complex CSCW systems.

Supporting software engineering covers a number of aspects [68], such as design, coding and verification. This exceeds the combined abilities of a whiteboard and a shared editor by far. The in [68] proposed tool TelSEE (Tele Software Engineering Editor), for example, covers the design aspect by providing a UML (Universal Modeling Language) [89] specific drawing

mode. In this mode a UML description of the system can be generated by a number of people working together. This includes version control as well as conflict avoidance (i.e., detecting and acting upon conflicting changes). Further this process should be supported by a session offering audio, video and whiteboard support as discussed above. For the actual coding of the UML specification a text based source code editor is available, again featuring version and conflict control. At any stage of the development, including the specification via UML as well as the textual coding modifications are propagated to all other team members. This allows all of them to view, evaluate and discuss these modifications instantaneously.

Automotive

The automotive sector offers a number of further specific examples regarding the usage of CSCW systems. While here again for meetings independent of the content the previously introduced audio, video and whiteboard combination will be used, more content specific tools will greatly improve the efficiency. Integrating, for example, CAD, simulation and planning tools into the system will allow to not only discuss required modifications in meetings, but to directly visualize the resulting changes to the design using the CAD program, even to run a number of short simulations (if the simulation process is sufficiently fast) and further to consider the changes introduced by the modification to the planning process. As each company uses very specific tools to do its design, simulation and planning these modules will be highly specific to the company under consideration, as discussed in the introductory scenario.

2.2 Example scenarios of using CSCW systems

Many different usage scenarios for CSCW systems exist. Therefore it is useful to identify a small number of typical scenarios used for further reference:

- Discussion group or work group
- Class room or lecture
- Podium discussion or press conference

CSCW systems which cater for these three scenarios must deal with enormously different requirements, thus, need to exhibit a lot of flexibility. To support these scenarios it is not sufficient to just adapt the GUI, as the whole system has to adapt in order to provide an adequate support. This includes all components, including the application modules.

In order to facilitate later discussions, especially considering control aspects and scalability it is useful to provide a detailed description of the three above mentioned scenarios. An even more detailed description of scenarios, their usage and especially their visualization can be found in [18] and [19].

2.2.1 Discussion Group

In this scenario a group of people is involved in a discussion, giving this scenario its name. They all have the same rights (thus only a single level of hierarchy exists) and their number is limited to only a few, somewhere around 10. They will typically be using audio, video for basic communication together with a whiteboard as a common drawing/sketching pad [13]. If bandwidth allows they may be using the media tools to simultaneously receive audio and video from all other participants. Figure 6 shows an example for such a layout of the video tool. Here all participants are represented by thumbnails. To enable easy identification of the

current speaker his thumbnail is displayed in the larger center panel (in this figure the MBone session "Places over the World" is displayed with such a layout).

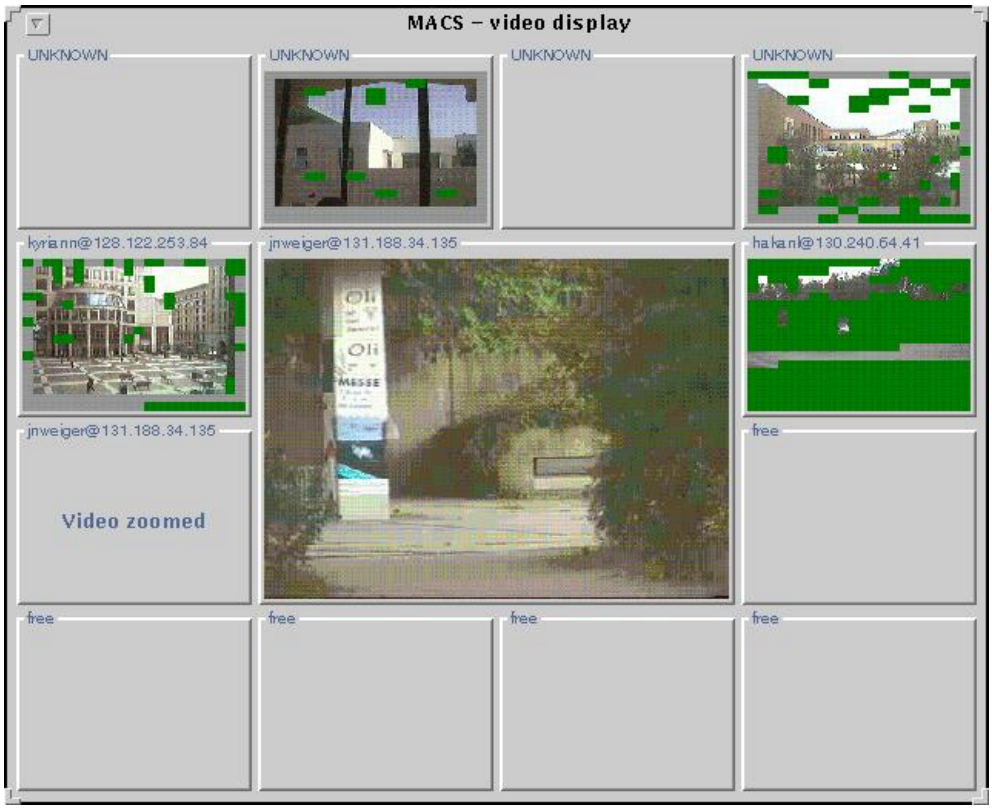


Figure 6: Example of video in discussion group scenario

Further the session participants might be using tools specialized to their work such as a CASE tool (e.g., TelSEE) for software engineers or in the automotive sector a shared CAD tool.

2.2.2 Class Room

This scenario models a normal class room with about 10 to 100 participants. One person, in general the teacher, owns extended rights and, thus, two levels of hierarchy exist. This type of scenario is often as well referred to as lecture scenario. Audio and video tools will, again, be used for basic communication. Here it is likely that the video of the teacher, or the object being discussed is displayed the whole time, while the others are more likely to send only on demand (in a typical setup most participants connection will not support all video feeds at the same time, actually with up to 100 participants hardly any existing system will manage). The whiteboard will mainly be used to present in advance prepared material [17]. Only if a question is raised the participants will be allowed to access, modify and add content to the whiteboard. For special subjects dedicated applications may exist. A virtual lab for performing physics, chemistry or biologic experiments comes to mind. The teacher will use something like a feedback tool to gain an estimation on how things are being perceived by the participants. Figure 7 shows an example of such a tool in use within the MACS framework providing ratings for received audio and video quality, as well as for the presentation speed.

Figure 7: Feedback tool in class room scenario

2.2.3 Podium Discussion

In a podium discussion or a press conference three levels of hierarchy exist. The moderator has unlimted rights and controls the session, the podium (e.g., a group of keynote speakers) has a set of limited rights, while the listeners (in comparison a very large group) have only enough rights to passively participate in the session or to submit a question to the moderator. Here the media tools are mainly used to broadcast the speakers audio and video. In general only a small number of people in comparison to the overall number of participants will raise questions. These will be collected by the moderator and then put forward to the panel. Only during his question will the participant transmit media data himself. The whiteboard will again be used to present previously prepared slide or to highlight aspects of a question. A voting tool might be in use, but it is somewhat unlikely that further tools will be used. The large number of participants results basically in a broadcasting setup mainly providing passive access.

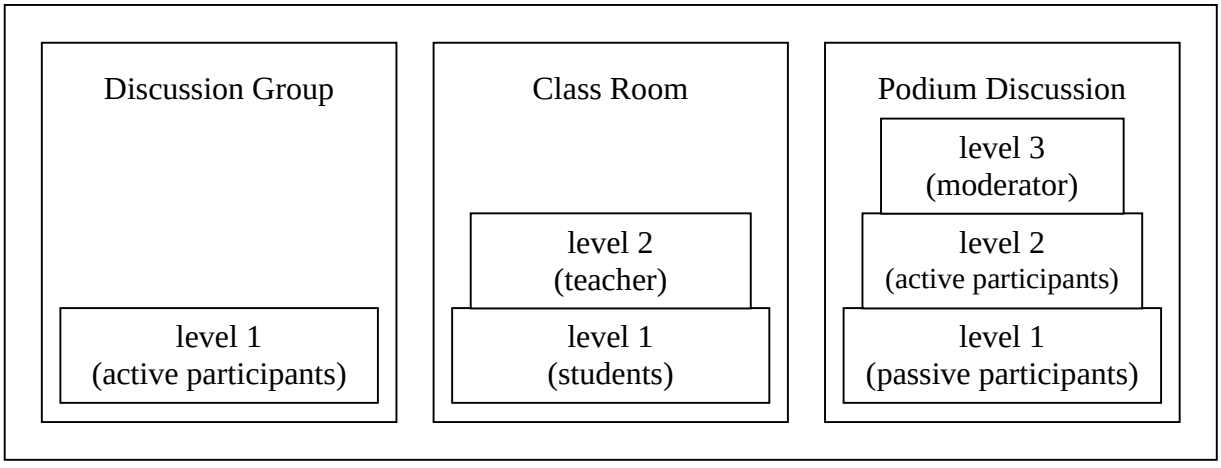


Figure 8: Comparison of scenario hierarchies

2.3 Control aspects of CSCW systems

While the application modules are the actual tools helping the user to work on a specific task, the control will determine the overall behavior and usability of the system. A CSCW system typically offers certain management functions through its control. These allow to access the sessions, users and applications known to the system. Further some limited configuration support is often provided. Here a modular design is of advantage, as it can be better configured to suite the needs at hand. Not just from the configuration aspect is modularity of advantage, but as well considering maintenance (well defined modules are easier to maintain than a huge monolithic systems) and resource usage (only modules currently needed are loaded thus preventing unnecessary resource allocation). An important special case is the configuration of the network support, a feature lacking in nearly all available systems. As a result an adaptation of the system based on the requirements of the specific usage scenario and the available environment resources (e.g., as offered by the network) is normally not possible. This is an important point, as especially with the development of active networks, as motivated in the introduction, new ways to handle and configure these and the applications using them are required. This type of adaptation should be part of the advanced configuration option of the system, not normally presented to the users. Only experts should access these. During normal system operation an automatic configuration and adaptation by the system is thus necessary.

Analyzing control aspects a bit more in detail a number of important issues can be identified, mainly concerning

- visualization of control,
- control strategies,
- control mechanisms and
- their corresponding protocols.

Each of these will differ according to the usage scenario. This document limits scenario specific considerations to the three previously introduced scenarios. Visualization aspects concerning the system control are treated. Session control visualization and control strategies for these sessions is outside the scope of this document (refer to [19] and [21] for more detail about these issues).

As a result only the system control visualization, the control mechanisms and the corresponding protocols used are further considered. Before continuing with these particular issues, it is useful to define some further session characteristics in order to facilitate discussion later on.

2.3.1 Visualization

Different aspects of visualization exist. These can relate to the selection of information displayed or the way the information is represented. Though the latter is an important issue it is not the focus of this document. Here only data about sessions, user and applications is of relevance. The visualization of the session itself is outside the scope of this document and treated in [19].

2.3.1.1 Visualization of session information

Before a session can be conducted, session setup has to be completed. In order to do so it is necessary to obtain and manage information regarding existing sessions. Therefore it is necessary to provide the following information and capabilities:

- list of available sessions
- access to session information
- create, join, leave and terminate sessions

Further the following issues need to be considered, but are as stated above outside the scope of this document:

- visualization of session and involved entities
- provision of floor control for session entities

2.3.1.2 Visualization of user information

User information must as well be gathered and visualized. The degree of detail is here even more dependent on the system under consideration than for the sessions. Many systems provide little user information, some none at all while not joined to a session. Systems supporting user list typically provide the following information and capabilities:

- a list of known users (if possible with information about their availability)
- access to more detailed user information
- mechanisms to invite and expel users (normally part of session management)
- a representation of communication relationships between users

2.3.1.3 Visualization of applications

Application visualization is mainly done by the application itself. Some systems choose to integrate control elements into their main window, as is often done, for example, for the volume control of audio module. A selector must exist to actually launch a specific application within a session. This functionality has to be provided by the control, as the application is at that stage not yet running. Thus, depending on the complexity of the system, the following issues arise:

- basic features
  - provide list of available applications
  - provide feedback about application usage (i.e., is a specific application used in the session?)
- advanced features
  - provide compatibility information (in order to allow different applications to work on same media)
  - provide version information

Even advanced systems do not necessarily support all of the above mentioned features, as for some systems these are of little relevance. Compatibility information, for example, is only relevant if the system allows the integration of different tools for the same media, such as two different audio tools. This type of information will typically not be directly displayed, but only available in advanced configuration panels or used to inform users if problems, such as, for example, incompatibilities occur.

2.3.2 Control mechanisms and protocols

Each CSCW system implements a number of control mechanisms and generally uses some protocol for executing these mechanisms across connected systems. Here, again, the extend of these functions varies greatly depending on the system under consideration. As a minimum mechanisms for the following functionality have to be provided:

- session setup (create)
- session termination (terminate)
- user integration into existing session (join and/or invite)
- removal of user from running session (leave and/or expel)

In general all these functionalities will comprise communication among at least two computer systems. A number of messages is used to define these functionalities and to inform or synchronize the remote system thus defining the protocol used. Here two possibilities exist, the system can use an existing protocol (e.g., SCCP [12]) or define its own. Using an existing protocol will be a first step to provide interworking capabilities with other systems, but a number of other issues have to be considered, as discussed in detail in chapter 5. The alternative is to define a proprietary protocol. This allows to customize the protocol for the specific system, allowing to chose the most efficient design possible. A disadvantage though is that certain standard functionality already defined in other protocols will have to be recreated. Despite this, as discussed in section 2.5, many systems use proprietary protocols.

Having discussed that some form of protocol will be used to exchange system/session information between involved computer systems it is necessary to consider the actual issues involved by the transmission of data among those systems. Often only a minimal network support is offered which typically is hard-coded to rely on a certain network infrastructure and network protocols. A flexible system allowing different network protocols being used must therefore provide the following:

- a number of transport services
- easy integration of new transport services
- manage network resources independently of the protocols (e.g., list of used addresses)
- allow access to advance transport service configuration

2.4 Scalability issues related to CSCW systems

In this section scalability with respect to CSCW systems is analyzed and discussed. First issues regarding scalability are investigated and different aspects of scalability defined. The following section analyzes the resulting implications on the components of a CSCW system.

2.4.1 Issues influencing scalability

Scalability typically refers to two issues, firstly the geographical separation between the participants and therefore the geographical distribution within the CSCW system, and secondly to quantitative considerations, here normally regarding the number of participants that can be supported without major performance degradation.

2.4.1.1 Geographical considerations

CSCW systems are often intended to allow a better integration of remote group members. Large companies are distributed throughout the world. Due to the ongoing globalization nowadays even smaller companies commonly have multiple locations or cooperation with remote partners. When considering CSCW system deployment in such environments scalability issues play an important role. Questions regarding the influence of great geographical distances between session participants on the performance of the system, its overall handling and stability have to be answered. Here the investigation of the network infrastructure and protocols used is most important. These considerations are not the focus of this work and are therefore not treated further explicitly, though some aspects are mentioned when analyzing quantitative scalability issues.

2.4.1.2 Quantitative considerations

Quantitative scalability considerations refer to the effects caused by a strong increase in the number of certain entities within a CSCW system. The most common entity considered is the number of participants taking part in a session, though the effects caused by others, such as the number of application modules involved in the session or the number of sessions joined should be equally considered. In the following section a number of issues is identified and discussed. Table 2 at the end of this section provides a summary about their effects on scalability.

Number of active CSCW systems

The reason to use a CSCW system is to cooperate with other people. Therefore each user must make his existence known. This can be done through mechanisms not integrated into the CSCW system (sending an email, placing a telephone call), but it is more convenient if the CSCW system itself provides an easy way of discovering the wanted contact information.

There are two basic ways of doing this. First one can use a well known service for exchanging contact information. This method requires a separate service to take care of the contact information, resulting in additional setup and maintenance requirements. Such systems can scale to support a very large number of users, as demonstrated by popular services like the Yahoo or Netscape web portals, or one of the many message services such as offered by AOL. The second possibility to make the local users contact information know is to broadcast this information directly into the network. While this approach does not need any additional setup or maintenance of a separate service, it does not scale well to larger numbers of users. With each user worldwide causing background traffic on the network and even worse, each user having to process all these incoming announcements, the network and the users systems will experience an unacceptable workload. Therefore this approach can only be used for small groups limited to specific network domains. Small refers here to the number of people involved.

Number of sessions

System resources available to the CSCW system are limited. For currently typical computer systems the main limitation are normally processing power, memory usage and network bandwidth. There are two additional resources which are relevant especially regarding usability. These are screen size [51] (i.e., the area available to display information to the user) and the ability of the user to manage information. A human being can only actively concentrate on one thing at a time, or passively follow a small number of simultaneous occurring events. Therefore it is not really possible for one user to simultaneously participate actively in more than one session. This will automatically limit the number of session a user will participate in. As a result the question of sessions per instance regarding scalability is of little importance. But one has to note that regarding visualization and access to unique resources, such as the systems audio and video subsystems the question of sessions per instance is very important.

Number of applications

A CSCW system should be useable in a wide variety of scenarios. To achieve this it is necessary to provide different applications to cater for the specific needs of a given scenario. As a result a number of different applications can simultaneously be active in a single instance of a CSCW system. In this case the same arguments apply as for the question regarding number of sessions per instance. Only here the screen space problem regarding visualization is even more prominent, as each application will tend to use an as big as possible part of the screen for itself in order to visualize its data.

Number of participants

Depending on the scenario chosen the number of participants can vary from only a few to a very large number. Further each scenario will require a different degree of interaction and coupling between the participants. This will have a drastic effect on the required processing power, memory and network bandwidth, as well as on the visualization. The number of participants per session is therefore one of the most important question to consider when regarding the scalability of a CSCW system.

number of	network usage	processing power	memory usage	user ability to manage	visualization	access to unique resources
active systems	-	-	-	=	=	n/a
sessions	=	=	=	+	+	+
applications	+	+	=	+	+	=
participants	=	+	+	+	+	-

Table 2: Categories and aspects of scalability

The scoring is defined as follows: - little relevance, = somewhat relevant, + relevant.

The effects caused by the increase of the number of a given entity (active systems, sessions, applications or participants) on the available resources is summarized and evaluated in Table 2. It is unfeasible for any non-flexible system to efficiently support small and huge groups alike. The system must provide a high degree of flexibility and adapt to the scenario chosen

by selecting an implementation optimized for that specific scenario. A number of typical scenarios modeling real world examples is defined in section 2.2.

Having a look at Table 2 a row at a time one finds, that the number of active and known systems will have little effect on the network usage, as these will only send small announcements in predetermined and fairly large time intervals (i.e., in the range of seconds to minutes) or register with some form of directory server. The processing power to parse these messages and update the database is on average again low due to the low frequency of requests and the fairly simple operation involved. Database size is given by the number of entities under consideration (e.g., participants) and thus will not grow to sizes were the search time for the entry will become significant. Similar the memory requirements are basically a linear function of the number of systems known and the size of the announcements. These are typically very small, especially in relation to today's typical available memory in computer systems. Only in the case of the directory server itself this will result in a clearly noticeable consumption of memory, as such a server potentially handles thousands of entries. Such a server is a dedicated system and not just a workstation or PC of a session participant. Thus, the server being dedicated for this specific task, it has no other processing intensive or real time relevant task (e.g., video coding/decoding) to do. Anyway, the setup of directory servers is not the point of interest, but its effects on scalability of a CSCW system using such a server. For a CSCW system ready to participate in a session or to create one by calling a communication partner the efficient visualization of a large number of potential communication partners/systems is a somewhat difficult task. Here a suitable presentation for the user is required in order to allow him to manage the potentially large amount of provided information. A purely alphabetical list of thousands of users is of little use and will be more a burden for the user than any real help. Access to system unique resources is only relevant within a session and therefore independent of the number of known systems. As a result it can be concluded that the number of active systems is only a minor issue for scalability.

The number of active sessions is on the other hand a more important factor. Here the network and memory usage, as well as the processing power required have a more noticeable effect. An active session results in a number of additional messages used to synchronize, for example, lists of participants. Most of these messages will typically be small, thus the main network, memory and processing power usage is caused by the applications. More problematic is the visualization aspect and the related question of the users ability to manage multiple sessions. The user will, in general, not be joined to multiple sessions at the same time, as normally one is not able to concentrate on multiple things at the same time. If at all, the user will most likely only be active in a single session and follow the others passively (e.g., just to get an overview of the content of these sessions). The visualization of multiple simultaneous sessions is difficult, as screen size is a limited resource and the interface should remain comprehensible [76]. A further important problem with multiple simultaneous sessions is the management of unique system resources. A simple example is the microphone or the video camera. For the captured sounds or video images a distribution to the different joined sessions must be achieved in such a way as not to disturb all other sessions with comments not meant for them. Thus, one can conclude that for the number of active sessions the main problem areas are the users ability to manage multiple simultaneous activities (i.e., session) and the possible conflicts regarding unique system resources.

The number of simultaneously active applications within the system is an even more problematic aspect regarding scalability. Here the main problems are the network usage and the required processing power. This will clearly depend on the specific applications in use, but assuming audio and especially video support much higher network usage and processing power requirements will result than caused by system or session information/synchronization

mechanisms. Memory requirements are somewhat relevant, but these depend again on the applications used and, thus, will only become significant for certain combinations of applications. Audio and video are played back in real time not requiring local storage (or only a comparatively small amount as a buffer). Other applications like a whiteboard will on the other hand require a lot of memory, causing an average overall memory consumption by the applications. The visualization aspects of - and the users ability to - manage multiple simultaneously active applications once again go hand in hand. The limited screen will force a selection of the representation of the applications and, thus, make the management by the user more difficult [22], as he can not gain a complete overview at a single glance. A point which is here of less importance than for the number of sessions is the usage of unique system resources. Applications are, in general, specifically geared to a certain task and thus will normally only use very specific resources. The microphone, for example, is very unlikely to be used by any but the audio application. Hence, for the number of active applications memory and processing requirements form the main aspects limiting scalability.

In general the most problematic issue regarding scalability is the number of participants the CSCW system needs to handle at the same time for any given session. The network usage caused by each participant himself due to being in the session is only somewhat relevant as the data volume caused by the control mechanisms in order to keep the participants in the session synchronized is quite small. Each session participant will, in general, activate a number of applications causing a data volume that easily may exceed the volume caused by the control mechanisms. This leads to the above described considerations of network and system load. The processing power will not only be affected by the number of application the participant activated, but as well alone by his presence. With an increasing number of participants the management of the internal databases and the overhead of the control mechanisms to ensure a consistent and correct view of the session state and the state of the involved participants will increase. This is equally true for the data needed to be stored about the participants and, thus, the memory needed by the system. Similar visualization becomes even more critical. The user is only able to efficiently manage a certain amount of detail, especially considering the limited screen size. Thus, an increase of the number of represented entities, in this case participants, will have to lead to reduction in detail in order not to over exercise the user. This again will lead to a less clear perception of the state of the session and its participants. An extreme example would be the representation of the participants in a sorted list for a huge conference with many hundreds or even thousands of participants. The user will not be able to manage this information effectively, making this specific visualization of the participants rather useless and, thus, a more sophisticated approach is required. The question of the access to unique system is not really relevant considering an increasing number of participants. Only in some cases very specialized resources might be affected, such as, for example, a remotely controllable camera. Such a camera allows the remote participants to look around at the place the camera is installed by sending movement commands to it. Here an increase of people sending possibly conflicting movement request will be somewhat relevant, but for nearly all other resources this is of little or not relevant at all. As a result the number of participants in a session mainly effects scalability due to processing power and memory requirements, as well as the users ability to manage large sessions and the systems ability to visualize these.

Overall one can see that for scalability the main point of interest is the number of participants in a session. These directly cause a certain load on the system and network (and hence increase resource requirements) by their pure presence in the session, but the main load on the system and the network is due to the applications started to process the media used in the session.

2.4.2 Components influencing scalability

This section analyzes scalability issues of the in section 2.1.3 introduced typical components of a CSCW system, except for the application modules. These contain their own very specific application code which will determine the scalability of the corresponding module and hence an analysis of the scalability has to be done separately for each given application module. Still many of the issues relevant for the whole CSCW system will equally apply. The main aspect of scalability considered here is the scalability with respect to the number of entities under consideration (e.g., participants of a session).

2.4.2.1 Communication services

The main point of a CSCW system is to enable cooperation. Therefore suitable communication services have to exist. These can range form the simple transmission of control and data objects (in form of a number of data packets) to much more complex systems offering Quality of Service (QoS) and other advanced services. Especially here scalability is an issue, as some of these advanced services are group orientated and will, in general, be directly influenced by the group size. Handling communication relations in a group is normally more difficult then just for a single instance. Especially if state information is required to track the status of the instances forming the group the efficient support becomes a complex task.

Still the basic to all this is the transmission of data packets. The implementation of this and the protocol used will have a strong effect on scalability. Based on the network infrastructure available at deployment further limitations may arise, allowing only the usage of some protocols, which might not be the most appropriate for the intended usage scenario. Ideally the underlying network will provide QoS guarantees to enable smooth operations of the CSCW system, but unfortunately this is commonly not the case. Section 3.2.3 discusses required network support and the topic of QoS in more detail.

The two most important factor regarding scalability are

- the type of communication used and
- the level of reliability required.

Types of communication

Four basic types of communication exist [101]. These are:

- **Unicast** (1:1)
- **Multicast** (1:n)
- **Concast** (m:1)
- **Multipoint** (m:n)

CSCW systems by their nature tend to use a Multipoint communication, where *m* is the number of senders and *n* the number of receivers. As Multipoint communication services are often not available they are emulated by Unicast and/or Multicast instead. A performance and scalability problem arises immediately if Unicast communications are used, as the data has to be send separately for each receiver or broadcasted. This places a high workload on the

sender, and in case of most computer networks (this is a typical schema from the telephony sector) this will additionally result in unnecessary usage of the network, as it is very likely that many receivers will have at least common subsections in the transport path used by the data packets. In this respect Multicast is a better solution. Here the data is send only once and distributed by the network itself [65]. As a result the data will normally traverse any part of the network only once. The disadvantage is, that the sender does not know or has any control over the receivers, as any system connected to the network can receive the data. For broadcasting purposes this is a suitable approach, as the sender is not interested in knowing the receivers, but for CSCW applications this is not the desired situation, as this leads to a number of management and security issues [74] [90].

The thematic of multicast transmissions has been an intense area of research over the last few years leading to a number of suggestions regarding security and reliability issues but these are outside the scope of this document.

Reliability

There are three different levels of reliability that may be provided by the transport services:

- reliable
- semi-reliable
- unreliable.

These service levels have a strong influence on scalability. The choice of the level of reliability is mainly based on the application and its individual requirements. For certain types of data the loss of some packets will be acceptable. Real time data, such as audio is a good example. Here the time to transmit the data (delay) is more important than the reliability (at least within certain limits). Delay considerations mainly apply to data from interactive processes and not that much to streaming data.

A reliable transmission guarantees that all receivers get all of the data, that it is correct and without duplications. According to the order in which the data is delivered further distinctions can be made in the case of group communication. To obtain such a guarantee it is necessary to receive an acknowledgment (ACK) from each receiver, thus, it is as well necessary to explicitly know each group member. If a receiver fails to receive the data or is separated from the sender (e.g., by a network failure) the sender will be directly informed. This level of reliability will cause the most network usage and processing overhead, as each receiver has to confirm each packet. Specific techniques such as local groups may help to reduce the overhead [49] [101].

A semi-reliable transmission does not require each receiver to confirm each packet. A typical approach is to use negative acknowledgments (NACK). The receiver can determine by the sequence number of a received packet if he did miss any previous packets and if necessary generate a NACK. The transmission system of the sender can then retransmit this packet. Only if it has the requested data no longer available it will inform the application of a transmission failure. To allow the sender to detect the separation from the receiver additional mechanisms are necessary. As a result the sender does not know the exact state of the receivers which can lead to inconsistencies between the sender and the receivers. This behavior is not acceptable for critical data (e.g., control databases). Data which can handle possible delays of transmission or delays in the detection of failure(s) can achieve a much

higher degree of scalability with the semi-reliable approach due to the reduced number of messages and, therefore, reduced processing power and network usage.

The unreliable transmission will simply send the data through the network without knowing if the receivers do actually receive the data. Further the receiver might get the data out of order, as some data packets might be routed via a different path. This requires no further processing by - or communication between - the sender and the receivers and allows therefore the highest scalability of the here presented approaches.

2.4.2.2 Control in CSCW systems

The control is responsible for the management of the CSCW system, especially for the sessions, users and applications. To fulfill this function two main aspects have to be considered: the management of the data describing these entities and the mechanisms used to excerpt control over these. Therefore the control can be subdivided from a functional view into the control database and control mechanisms. Here the in Table 2 identified aspects of scalability (except user ability to manage) can be associated as follows:

- Control Database
  - processing power
  - memory usage
- Control Mechanisms
  - network usage
  - visualization
  - access to unique resources
  - degree of coupling (not shown in Table 2)

Control database

The control database contains information about the sessions, applications and users. In a tightly coupled system all users must be known and information about them stored in the database. This clearly limits scalability due to two factors: The resource requirements to store the data will become very large with increasing number of participants, though modern systems might even be able to provide the required resources. A much more serious problem is the overhead needed to keep the information up to date. In a large group there will be a lot of fluctuation resulting in a massive overhead just to process requests and to update the database (group dynamics). Especially at critical times, such as start and end of a session, the number of requests will cause severe processing problems. A loosely coupled system needs less data and has not as stringent requirements for data updates. This will result in a less comfortable system for the user, as he might work with limited or not up-to-date information about other users, but allows a higher degree of scalability. Thus, due to scalability considerations it is necessary to keep the number of tightly coupled participants in a group as small as possible, while still allowing a large overall number of participants.

Control mechanisms

The control offers a variety of control mechanisms depending on the session type and usage scenario (as defined in section 2.2). Control mechanisms are responsible for translating events such as, for example, a person joining the session, a person leaving the session, a transfer of rights or a session termination into a certain sequence of actions. Taking a person joining a

session as an example the following applies. The person wanting to join the session selects the specific session from a list of sessions and presses the join button (i.e., visualization). The expected result is that he will be joined to the session. If this fails he will receive an error indication. To achieve this the join mechanism is started leading to a sequence of actions, such as checking available resources, determining the session chair, determining the session type and depending on it asking the session chair for permission. As a result an open session will be more scaleable than a moderated session since the overhead, for example, is smaller considering the session chair.

Independent of the session type there are certain phases in the lifecycle of a session which will have a more severe influence on the possible scalability. Especially the session start is a critical phase, as a large number of participants will issue nearly simultaneous join requests, leading to a substantial processing load for the session controller. For the session chair in a large session it will be difficult to handle such a large number of requests. Most other operations (e.g., expel) are less critical and will occur in more uniform distributed manner. As a result a distribution of workload (and as well network usage) caused by the control mechanisms as qualitatively shown in Figure 9 can be expected, with a high load during the initial starting phase caused by many incoming join requests and a smaller increase in load again towards the end of the session by the processing of an increased number of leave requests.

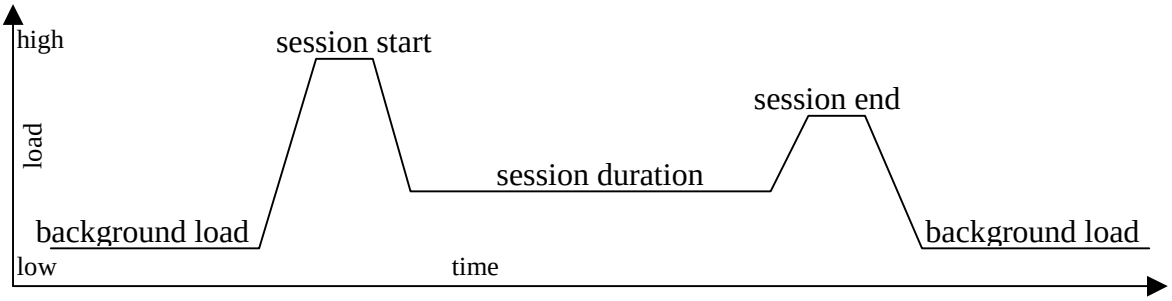


Figure 9: Expected distribution of load caused by control mechanisms

2.4.2.3 Graphical User Interface

For a collaboration system the aspect of visualization is of extreme importance, as it is one of the main factors determining usability. A concise presentation of each user within the session management tool will help to raise group awareness and therefore improve collaboration. With increasing number of participants such a presentation becomes more difficult, as screen space is limited, as well as the users ability to manage efficiently large amounts of information. Therefore the detailed presentation of each participant of a large group is not efficient or even possible. The topics of visualization and user perception are treated in detail in [19] [20] and therefore not explored further in this document.

2.5 State of the art of CSCW systems

A steadily increasing number of CSCW systems exist. These can roughly be divided into the following categories:

- Web based systems,
- ITU H323/T120 based systems,
- systems using the MBone and
- proprietary developments.

Each of the following sections will discuss the general advantages and disadvantages of these categories and introduce some representative examples A summary considering advantages and disadvantages will conclude this section.

2.5.1 Web based systems

Web based systems have a number of advantages. Normally they only require a Java enabled browser to operate and thus no user installation is required. This is not only the easiest way for the users to gain access to a CSCW system, but is as well very attractive for companies [30], as they can centralize their service and maintenance [37]. As Java applets, in general, are only allowed to establish network connections with the server they have been loaded from (due to security considerations) these systems will normally be server centric. This can be a disadvantage due to scalability considerations. Further disadvantages are based on the applet running in a browser. Java is, in general, quite slow and the browser environment will further limit the applets. These limitations are partially based on the security restrictions inherent to Java applets. A further problem is the access to local hardware, such as audio and video devices. Java support for this is provided by the Java Media Framework (JMF) [94]. Unfortunately at this stage of development of JMF it is still far away from being really useable or even stable. Version 2.0 finally allows audio and video capture, but only a very limited set of formats is supported and operating system integration is minimal. The main point of additional Java packages required by a Web based CSCW systems is that these will require user installation and therefore the advantages regarding no user installations and maintenance are lost.

2.5.1.1 BSCW

The web based CSCW system BSCW (Basic Support for Cooperative Work) was developed by GMD in Germany. It uses plain HTML and dynamically generates web pages. Plain text messages and list of contacts/addresses are provided, as well as download links for objects such as documents. The main aspect is the exchange of documents via a web based server. Version information and access lists are kept to provide a better feedback of the status to the users. It is as well possible to setup meetings and store contact information. The developers of the system see BSCW as an enabling technology [7]. It has minimal requirements, but offers the basic support of cooperative (asynchronous) work. Video conferencing or real time shared applications are not supported, same as synchronous cooperative work in general. Therefore this system is not regarded any further. Table 3 summarizes the capabilities of BSCW.

portability	**flexibility**	**session control**	**scalability**	**interworking**
+ / good, only HTML compatible browse required	- / bad, mainly limited to document exchange	- / not applicable, no sessions available	- / not applicable, no session available	- / not applicable, no session available

Table 3: Capabilities of BSCW

2.5.1.2 JETS

JETS (Java-Enabled TeleCollaboration System) is a typical representative of web based systems. The system is developed by the Department of Electrical and Computer Engineering at the University of Ottwa, Canada. The current version is called JETS 2000. It is a client-server model using Java applets on the client side aimed at supporting synchronous cooperative work. The system supports session management and provides some sample application modules such as a whiteboard and a VRML viewer. A more detailed description of JETS and some remarks regarding the use of a client-server / Java applet model to implement a CSCW system are given in [38]. They as well point out that JIT (Just In Time compiler) and HotSpot [93] technologies help to minimize the Java speed disadvantage and claim that speed close to native code, even in the browser environment are possible. The problem of audio and video support are by them as well recognized and marked for further investigation. For playback a H.263 compliant decoder was implemented [37]. Session management is mainly limited to session setup. Access can be restricted via passwords. The session chair has a list of joined and invited (but not yet joined) users. Further he can display various system state information, as, for example, a per user list of applications in use. Basic floor control is implemented by assigning permissions to each application. These can prevent the user from starting a given application, provide him with read-only access or allow complete (i.e., read and write) access. The API for creating custom applets usable with the JETS system is available and, thus, extensions to the existing system are possible. The capabilities of JETS are summarized in Table 4.

portability	**flexibility**	**session control**	**scalability**	**interworking**
+ / good, based on Java applets	= / average, only extension by custom application modules possible	= / average, basic control and limited floor control available	- / bad, server centric and single control model for all group sizes	- / not applicable, not available

Table 4: Capabilities of JETS

2.5.1.3 Tango

In contrast to JETS and BSCW requires Tango the user to perform local installations, in this case a browser plug-in. Otherwise all comments stated above regarding web based CSCW systems apply. Tango does cover some additional aspects. Besides synchronous cooperative work asynchronous operation is supported via a database backend [4]. Further, provisions are included to handle distribution of decentralized media streams under the control of Tango. Tango has been commercialized and is used in a number of setups. Integration of non-Java code is supported, at the prize of further local installations. Due to long load times during session setup, provisions for a version of Tango allowing for a local installations are underway. Tango features the usual session setup and access control. Data and events are strictly separated in Tango. The system itself only handles the events while data is always directly obtained by the application from its server or connected database. Session management is done through an applet, the SM (Session Manager). It allows to access the user and application list, showing in the later the currently active applications. Multiple instances of an application with different user subsets are possible within a single session, an approach not commonly found in other systems. A further interesting approach is that

applications can exist independent of sessions [5], thus not only users can join and leave sessions, but applications can behave in a similar way.

Overall does Tango offer a large amount of functionality including some interesting and uncommon approaches. A variety of application modules are available. Tango does require local installations (plug-in plus non-Java application modules) and, thus, does not deliver the typical advantages of web based systems with respect to maintenance and portability. The capabilities of Tango are summarized in Table 5.

portability	**flexibility**	**session control**	**scalability**	**interworking**
+ / good, based on Java applets, but requires local plug-in	= / average, only extension by custom application modules possible	= / average, basic control and limited floor control available	- / bad, server centric and single control model for all group sizes	ITU interworking (NetMeeting) supported

Table 5: Capabilities of Tango

2.5.2 ITU H.323/T.120 based systems

The H.323 and T.120 standard families of the International Telecommunication Union (ITU) describe the structure of - and the protocols used for - multimedia conferencing systems. A more detailed description is given in section 5.1.1 in the context of interworking between MACS and NetMeeting. The main advantage of ITU based systems is their interoperability. Based on a company independent standard the products of different companies can work together allowing a better uptake of the technology. These standards are well developed and have obtained a wide recognition. The disadvantages are their high resource requirements in terms of manpower to implement and computer resources to run. Further, they originate from the telecommunication area, and tend to support typical telecommunication network structures better than computer network structures [48] (e.g., very limited multicast support). Many companies selling ITU based systems have integrated further (useful) tools not defined by these standards. These are generally not interoperable with other systems, limiting interoperability to basic protocols.

2.5.2.1 NetMeeting

NetMeeting is a CSCW system based on the ITU H323/T120 standard families. It is freely available from Microsoft and runs on their Windows operating systems. It uses H323 to implement audio and video connectivity. The address of the wanted communication partner can be entered directly or found via a search of a publicly accessible directory, such as *ils.microsoft.com* or *ils.four11.com*. These directory services are based on X.500 [62] and accessed via LDAP (Lightweight Directory Access Protocol) [50]. While version 2 of NetMeeting used these by default for selecting participants, version 3 seams to aim by its user interface and default configuration much at video-telephony than at videoconferencing. Further, NetMeeting does offer chat, whiteboard and file transfer functionality. The file transfer is T.120 conform. Application sharing is supported as well. This is achieved by duplicating the application window and displaying it for each participant individually. The advantages of such an approach are that the participating users do not need to install the application themselves, actually the application is not even aware of it being shared. On the other hand that is exactly the point causing problems. A fine grained access control is not

possible, once the control is transferred to a given participant he can then use the application on the owners computer without any limitations. In an extreme case this could mean that by sharing an Explorer it is possible to format the owners hard drive. A much less dangerous, but more problematic point during normal usage is, that the window is shared with the exact same dimensions and position. For a user with a smaller screen (as in number of pixels) parts of the window will not be visible, without the owner of the window actually noticing this problem. While this form of application sharing is a simple and efficient way for showing others work under consideration and to request comments it does not allow to work simultaneously and to cooperate efficiently.

Overall does NetMeeting offer a fairly nice, but very limited support for CSCW. The main limiting factors are hereby portability and scalability. The system is bound to the MS-Windows operating system and by its user interface and default configuration basically aims at supporting two people communicating with each other. It does allow cooperative work via application sharing with some limitations. Table 6 shows again the capabilities of NetMeeting.

portability	**flexibility**	**session control**	**scalability**	**interworking**
- / bad, based on MS-Windows	- / bad, extensions are very limited and not easily possible	- / bad, only very limited functionality	- / bad, limited by H.323 and T.120	- / not applicable, only with other ITU based systems

Table 6: Capabilities of NetMeeting

2.5.3 Systems using the MBone

The MBone is a virtual network on top of the Internet. It is described in more detail in section 5.1.2 in the context of interworking between MACS and typical MBone tools. MBone itself does not define any application, system structure or even protocols used within a CSCW system. On the other hand some protocols are typically used on the MBone. With the spread of the MBone some tools did gain a wide acceptance typically implementing these protocols. Those tools are now in a number of projects refined, extended or ported to other operating systems. However, they were initially just developed for experimental purposes and, thus, should not be viewed as fully-fledged video conferencing or CSCW systems.

The main advantage of MBone based tools is their ability to efficiently distribute data due to the inherent multicast capabilities. This results in high scalability. The disadvantage is the network infrastructure required. While in principal any Internet host can be connected to the MBone, in general, this will not be the default setup and hence has to be manually done by the user or the system administrators. As tools used on the MBone vary greatly their specific advantages and disadvantages have to be discussed individually.

2.5.3.1 MBone tools (*sdr, vic, vat* and *wb*)

As MBone is more an infrastructure implementing a certain technology it is not possible to refer to any tool as the MBone tool. But certain tools were developed right along with the MBone and are known to many people familiar with the MBone. These are, for example, *sdr*, *vic*, *vat* and *wb* [65]. Used together they form a system useful for simple videoconferencing or

broadcasting of video. Examples for the latter are the NASA transmitting some of their space shuttle missions via MBone, as well as FauTV in Germany, transmitting a TV channel.

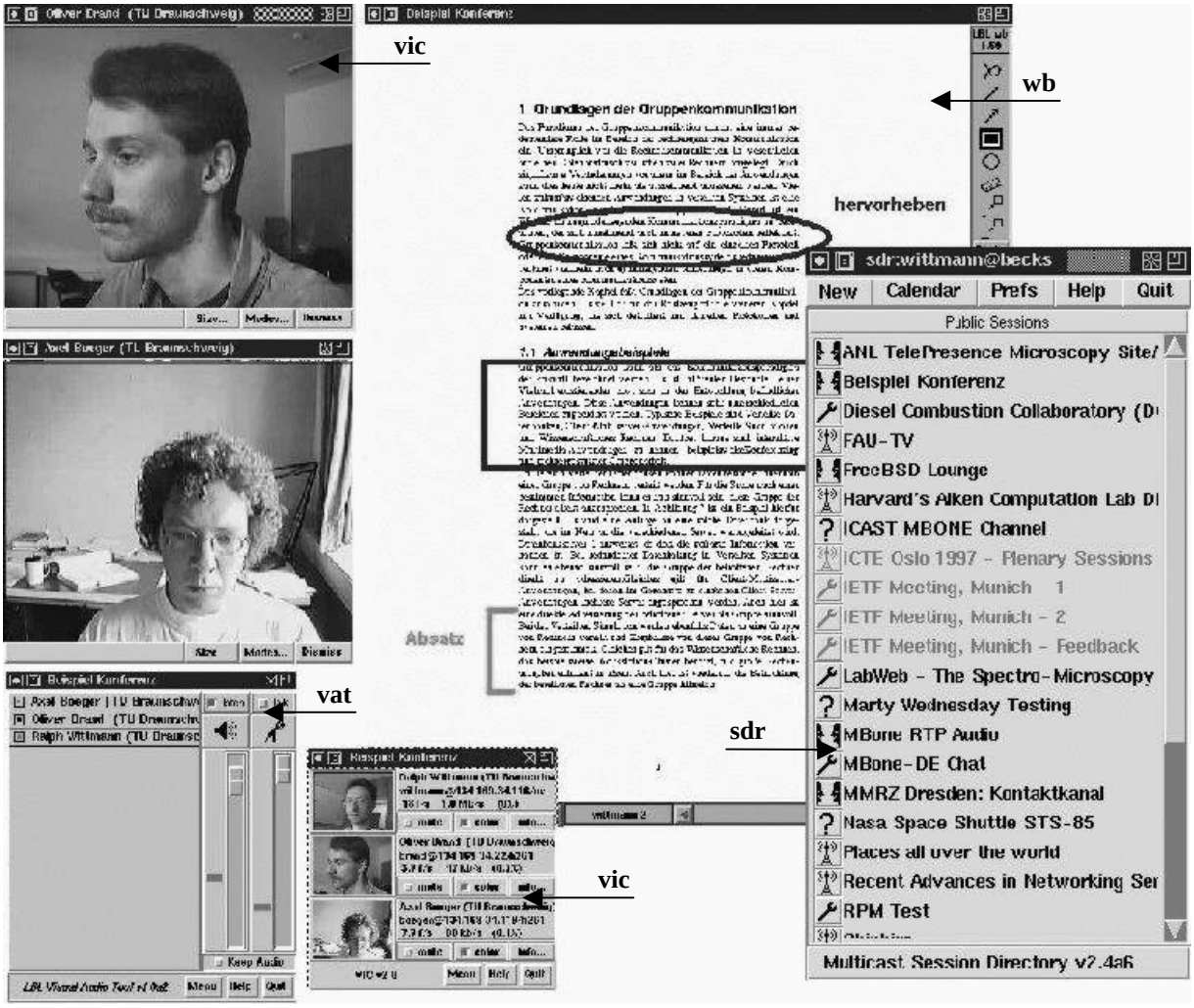


Figure 10: MBone tools

sdr is a session directory tool, allowing to receive and generate session announcements. It as well functions as a launcher for the other tools. *vic* is a video tool allowing the playback off multiple video streams and the capture of the local video, if supported hardware exists. *vat* provides audio functionality. *wb* is a whiteboard allowing slides to be transmitted and provides shared drawing capabilities. Figure 10 shows the above described setup.

A number of projects are concerned with extending or porting these tools, or implementing tools with improved functionality. One example is the CDT mStar project [81]. This project aims at developing a suite of tools similar to the above mentioned by using as far as possible Java as an implementation language. This will allow their use on any platform supporting Java, or at least of some of the components, while reducing the porting overhead for the others. Besides the session management, audio, video and whiteboard they as well support a voting tool.

For all these tools it applies that they in principal offer a high degree of flexibility and scalability. Each tool has a specific function and can be easily replaced by a better implementation with the same function if available. The tools offer little access control. To enforce privacy some support encryption. The MBone is an area of ongoing development.

Using the MBone it is currently not easily possible to conduct a true private session. A further disadvantage is the loose coupling of the tools, providing no default synchronization between the tools (e.g., audio, video and whiteboard synchronization). In Table 7 the capabilities of the above presented MBone tools are summarized.

portability	**flexibility**	**session control**	**scalability**	**interworking**
- / bad, except for mStar project	+ / good, easily exchangeable and extensible	- / bad, normally no control within session available	+ / good, efficient data distribution	- / not applicable, no session control but often compatible coding used

Table 7: Capabilities of MBone tools

2.5.3.2 MBUS

To improve the shortcomings of typical MBone tools, as those introduced above, especially considering the aspect of insufficient coupling of the tools one approach suggested is MBUS. MBUS is a message bus for coordination of local applications. This assumes a typical MBone setup with separate tools for each media. To allow control mechanisms, that is, for example, to group applications into sessions, it is necessary that these applications exchange control data. To achieve this the MBUS sets up a local multicast group to send messages between applications. The message types range from informational indications to requests, which require the receiving application to confirm the message. Messages contain a user key to prevent reception of messages belonging to a different user. This can happen, as the local multicast group will be shared with other users on the same system. While, in general, system refers to a single workstation this is no build-in limitation. Thus to actually run the audio tool on a different workstation than the video tool is quite possible. Further, messages use authentication and encryption to prevent (malicious) interference by others. The MBUS was submitted as an Internet draft [79]. MEGACO is a draft for applications on major telecommunication equipment and in part covers similar aspects as MBUS. MBUS capabilities are summarized in Table 8.

portability	**flexibility**	**session control**	**scalability**	**interworking**
- / bad, as extension to existing MBone tools	+ / good, same as for MBone tools	+ / good, session control possible, extend depends on each application	? / unknown, depends on application, not yet known	- / not applicable, same as for MBone tools

Table 8: MBUS capabilities

2.5.4 Proprietary developments

A number of proprietary developments exist which are not ITU, MBone or web based. The advantage when designing a system using proprietary standards is that one can choose structure and protocols freely to best suit ones needs. The disadvantage is the high development overhead, as certain problems typically solved in existing standards have to be solved and implemented again. Further is the interoperability limited, except if special interworking code or modules are included.

2.5.4.1 Habanero

Habanero implements a CSCW system written in Java. It therefore possess the before mentioned advantages and disadvantages typical to Java based systems. Though Habanero is not a web based system it uses a client-server structure. The server possess an arbitrator to distribute executable data objects [29] to all clients, enforce event ordering and floor control. Java serialization is used in the transmission of objects to the clients. As these objects can be anything from a mouse click to a complete program this feature provides an enormous potential, but as well a source for many problems. Java serialization is still under development and differs from version 1 to version 2 substantially, with further changes announced [26] [36]. The API is documented and the integration of new applications (called Hablets) into the system can be done via a text based configuration file. Quite a number of Hablets exist, including the usual whiteboard, chat and voting tools. Further a shared file system, FTP like file transfer and a network browser (like the Microsoft Network in Windows) are provided as general purpose tools. A number of special purpose tools and demonstrations, mainly from the fields of physics and geographics exists, as well as some simple games. Standard session control is offered in a single integrated window, as shown in Figure 11.

Overall Habanero offers good functionality and a wide variety of tools and demonstrations, more so than any of the previously considered systems. The for a Java system typical lack or at least limitations in the audio and video support are as well present in Habanero. Table 9 summarizes the capabilities of Habanero.

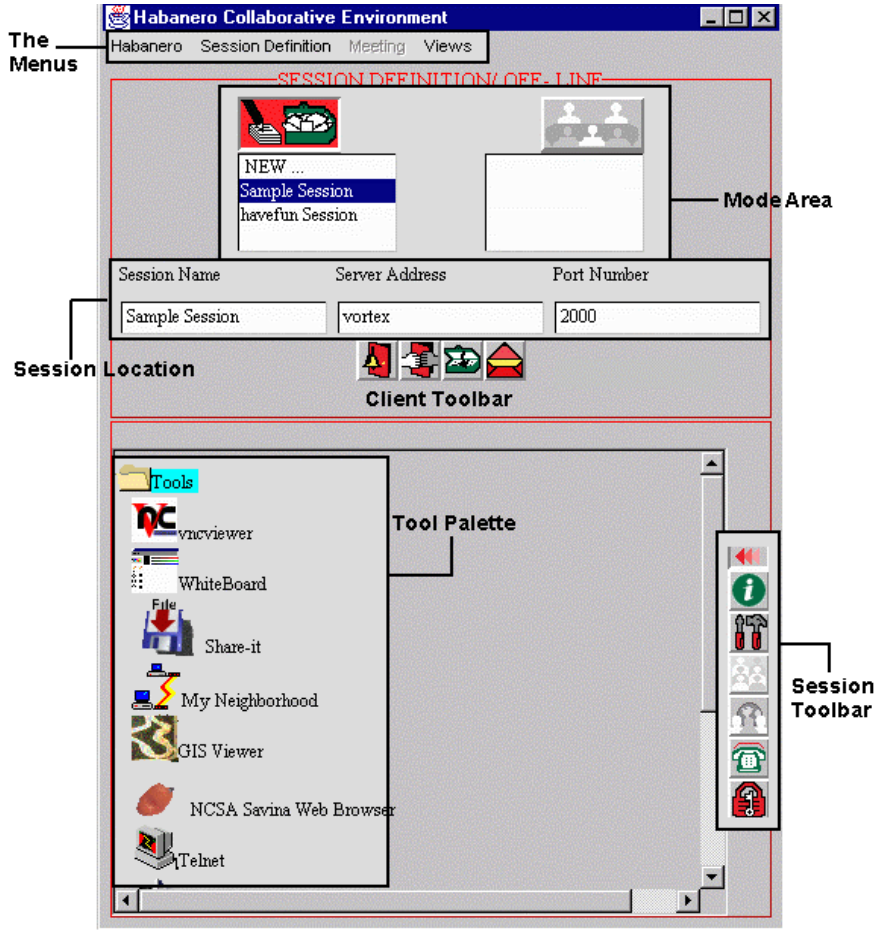


Figure 11: Habanero session control

portability	flexibility	session control	scalability	interworking
+ / good, Java based	+ / good, various setups supported and extension via Hablets	= / average, basic session control	= / average, different setups possible, allowing adaptation to usage scenario	- / not applicable, not available

Table 9: Habanero capabilities

2.5.4.2 IRI

The IRI (Interactive Remote Instruction) system is a development by the Old Dominion University. It has been used a number of times since fall 1995 and developments are ongoing. The system uses both reliable and unreliable multicast and contains components (e.g., gateways) for deploying such infrastructures. Further, a heavy reliance on third party products (commercial and others) exists in order to deal with problems such as audio, video and application sharing. Network requirements are defined and usage is normally expected to take place in well engineered Intranets supporting these specifications (e.g., 2MB/s, RTT 50ms, max. 3% loss rate).

Situational awareness is created by a shared workspace containing all media used in a virtual room, as shown in Figure 12. This ensures that all participants have the same view, but this approach as well requires a large amount of screen space and may in certain cases be somewhat inflexible. Application sharing is supported (implemented via a commercial tool) for certain setups and uses the same principals as the application sharing of NetMeeting. A session control is available and a token management to regulate access is implemented. Both are mainly suitable for trained professionals and not so for novice users. The capabilities of IRI are again shown in Table 10.

Currently a new follow-up system called IRI-h is under development, aiming at improving some of the problem areas detected in the current system. This includes, for example, more flexibility and portability by moving to a partial Java implementation.

Figure 12: IRI workspace

portability	flexibility	session control	scalability	interworking
- / bad, dependent on not portable third party products	- / bad, no easy customization	= / average, basic session control and token management	= / average, support for different setups available	- / not applicable, not available

Table 10: IRI capabilities

2.5.4.3 CLear

CLear is a system developed by GMD-IPSI, Darmstadt, Germany. It focuses on Cooperative Learning and places a high importance on the structuring of information and relations [6]. The system is implemented in VisualWorks Smalltalk and, therefore, can run on all platforms supported by it. CLear uses the metaphor of rooms. Each user has his own room where he can prepare, modify and store hypermedia information. The user has the control over all objects in his room, even if others are allowed to enter the room or are invited. This is equivalent to a closed session controlled by the creator. Further types of rooms exist, these can be libraries, used for information storage, group rooms used for discussions and auditoriums used for lectures. This structure is visualized in an explorer like tree and, therefore, easy to understand and use. Users are visualized by thumbnails located in a column at the left side. The cursors of these users are represented by a smaller thumbnail with an arrow attached to it. These allows

the immediate identification of the user acting, for example, on a whiteboard slide. As the structuring of learning is important in this project a protocol has been defined to support the learning process [73]. A simple example would be a script like implementation of an explanation process. If user A asks a question to user B, B will try to answer. In real life B will use hints such as the facial expression to evaluate if his explanation is useful to A or if he has to rephrase it or expand on it. With the computer these type of feedback are commonly not available and have to be provided in a different manner. While the concepts of rooms and the thumbnail support make for an easy to use system, they at the same time limit the scalability drastically. Imagine a setup with many users (dozens to hundreds). The tree visualizing the user rooms, group rooms and auditoriums will have many hundreds entries making it virtually unusable. The same will apply for actually working in large groups or in an auditorium. Filled with hundreds of thumbnails it is questionable if actually working on a document or a specific object will remain possible. Table 11 summarizes the CLear capabilities.

portability	**flexibility**	**session control**	**scalability**	**interworking**
- / bad, dependent on specific environment	- / bad, no easy customization	= / average, basic session control	- / bad, definitely limited by visualization	- / not applicable, not available

Table 11: CLear capabilities

2.5.5 Summary

Table 12 summarizes the provided features of the state of the art systems as introduced. For each of the categories and for each of the system in this category the evaluation from the system specific tables (Table 3 up to Table 11) have been summarized in Table 12. Further the features of MACS have as well been added.

	Web based			ITU based	MBone based		Others			
	BSCW	JETS	Tango	NetMeeting	M.Tools	MBUS	Habanero	IRI	CLear	MACS
portability	+	+	+	-	-	-	+	-	-	+
flexibility	-	=	=	-	+	+	+	-	-	+
session control	-	=	=	-	-	+	=	=	=	+
scalability	-	-	-	-	+	?	=	=	-	+
interworking	-	-	ITU	-	-	-	-	-	-	ITU MBone

Table 12: State of the art feature overview

As can be seen no system currently available fulfills all of the features identified as important for CSCW systems in this chapter. This did lead to the design of MACS. The next chapter describes the design of MACS, showing how it supports each of these features.

3 Design of the MACS system

MACS (Modular Advanced Collaboration System) provides not only a flexible, scalable and portable framework, but as well an implementation of basic tools. Various scenarios like the discussion group or class room are directly supported and others can be easily integrated. Interworking modules allow MACS to interwork with MBone or ITU-H.323 based systems. Adaptation to specific usage scenarios is possible due to MACS high degree of flexibility, modularity and the extensive configuration possibilities. At the same time MACS pays special attention to maintain an easy to use system. The following sections will first introduce the basic structure of the MACS framework, then proceed to discuss each of the in the following section identified levels in more detail.

3.1 Basic Structure of MACS

In order to achieve flexibility a highly modular approach is needed [83]. This is best realized if a clear structure is defined for the complete system. MACS is therefore divided into four levels, as shown in Figure 13. These are:

- Network Services
- Tools
- Control
- GUI

Each of these levels provides a well defined functionality to other levels. Modules within these levels can therefore be substituted, without need for changes to the rest of the system.

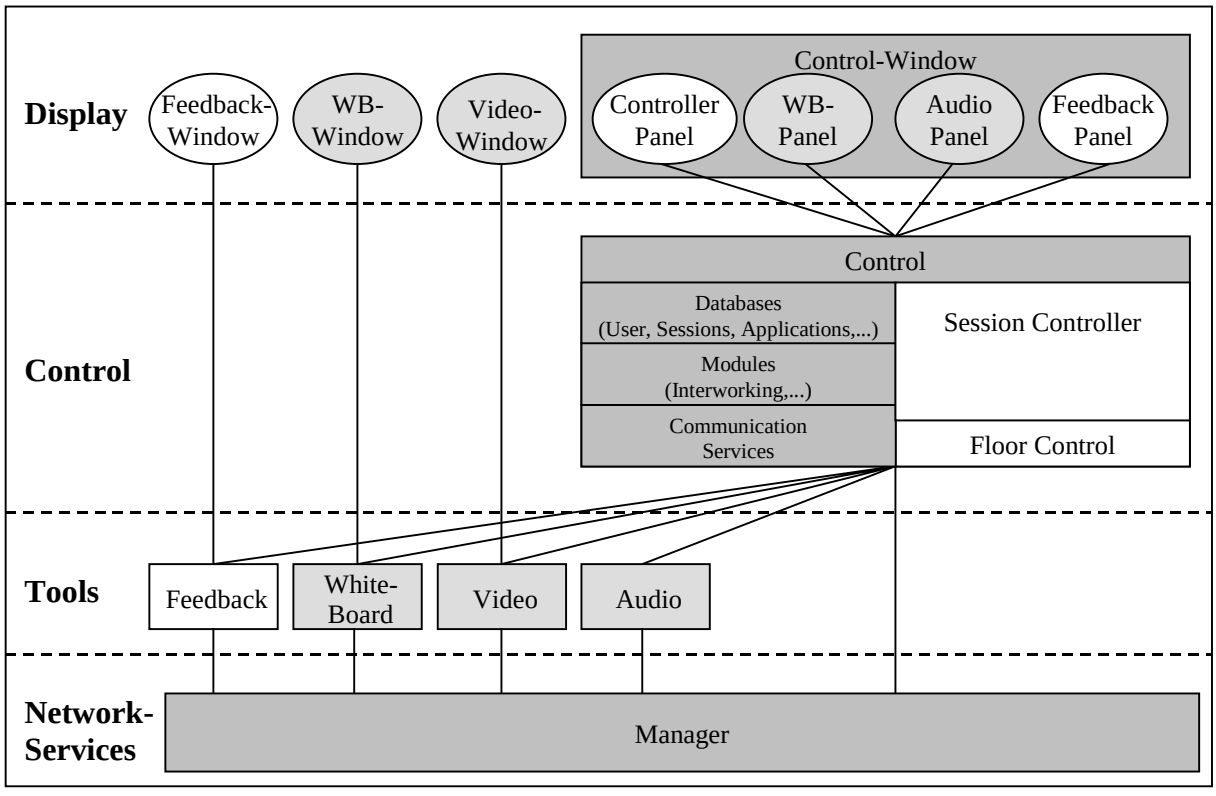


Figure 13: MACS structure overview

The above figure, while giving a good overview of the MACS structure, is in now way complete, as each level can be further subdivided into more detailed structures. The shown applications are just examples, other applications do exist. The main area of focus of this document is marked in the figure by the shaded components. For details about the session controller, floor control or the controller panel it is best to refer to [19].

3.2 Network services in MACS

Network services, as defined in section 2.1.3 must at least provide simple data transmission capabilities, but may offer various enhanced services. This section discusses the MACS network services, by first analyzing the requirements by the MACS framework and then discussing the resulting design goals. This is followed by scalability considerations relevant to the network services. Finally the structure of the network services level is described in detail followed by an evaluation of the here introduced design.

3.2.1 Requirements for network services

The interface to the network is very important, as it must isolate the system from the details of the underlying network technology and protocols, but at the same time it must allow a wide variety of implementations. The performance of the network level will still be determined by the underlying implementation which is the choice of the programmer or user, but the system will be able to use a different implementation (with different performance) without any changes to the code. What is required to achieve these aims?

- The system must be independent of network infrastructure and protocols.
- Deployment of environment specific network support modules is required.
- Group communication capabilities must be provided.

The last point is the most basic, as for any CSCW system the main point of interest is the collaboration within a group, which has communication capabilities as a prerequisite. It is still important to explicitly note this point, as the underlying network may not support these kind of services directly, making a service implementation on top of the available infrastructure necessary. An example is a normal TCP based network. Here group communication is not directly possible and therefore has to be explicitly implemented. This will result in the setup of a number of TCP connections to distribute the data, as requested, to a group of systems.

3.2.2 Design goals of network services

The design must hide the network specific details from the system. Further, support for different employment environments must be provided by defining, or at least allowing, a number of different modules implementing the basic network service functionality. This will cater for the specific network infrastructure and protocols of the chosen deployment environment. As a result some form of management instance will be necessary which will provide a form of translation service between network specific details (e.g., for the addresses) and a network independent representation within the framework. Thus the following design goals can be identified:

- network specific details implemented by modules
- modules offer general interface to system
- management instance provides translation/mapping services

The management instance must as well be able to handle network related system resources in a unified way in order to prevent conflicts. These could be caused by requests made by different network specific modules operating at the same time. An example would be a reservation for a specific port by two different modules, one, for example, trying to establish a socket for a TCP connection and the other for a RTP stream. Therefore not only is a mapping between the network independent representation by the system and the network depended representation by the network module necessary, but internally a list of used system resources must be managed.

3.2.3 Scalability aspects

Isolating the system from the network specific infrastructure also isolates the system from the network specific features. Group communication services are expected by the system and are provided by the network services. The actual implementation though will have a fundamental impact on the scalability. If the underlying network infrastructure, for example, provides multicast a much more efficient implementation is possible than on a TCP based infrastructure forcing the implementation to establish various TCP connections in order to distribute the data to all group members. Therefore scalability aspects have to be considered separately for each specific implementation.

3.2.4 Structure of the network service level

Considering basic functionality and the design goals stated above, the network services must have the following internal structure.

A management instance has to exist in order to provide the mapping between network independent and dependent representations. A number of modules containing network specific implementations have to be provided and managed. This includes the actual transmission functionality, as well as network resource relevant functionality. The later mainly refers to the management of addresses specifying the corresponding connections. The network service level, as shown in Figure 13, can therefore be further subdivided as shown in Figure 14.

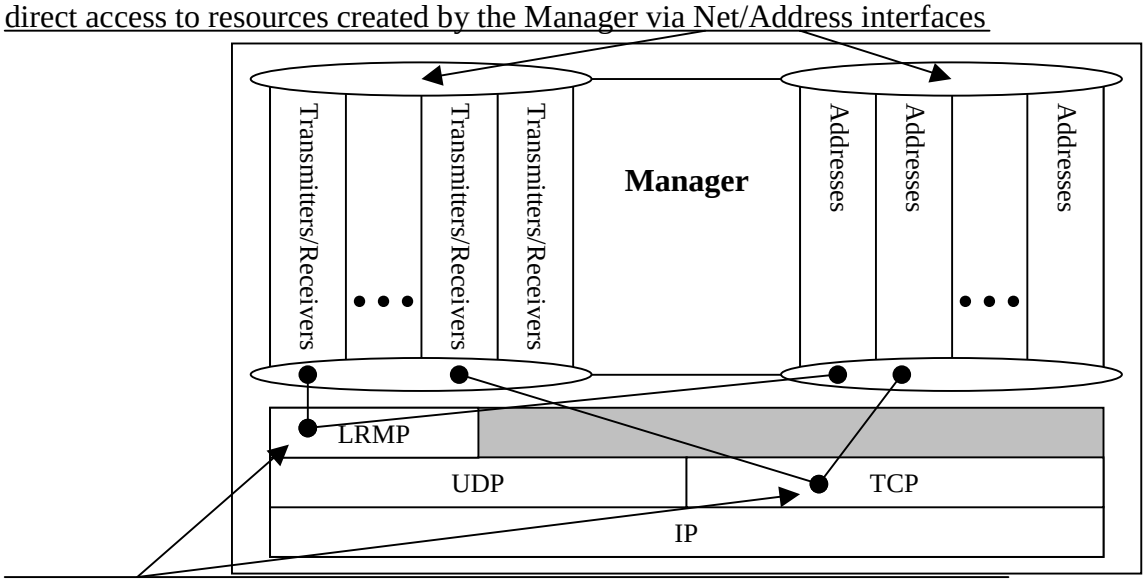


Figure 14: Detailed structure of network services

The Manager manages the transmitter/receivers and their corresponding addresses. Further other addresses received via session announcements, that is addresses used in not locally managed sessions, can be entered into the managers address list in order to prevent local allocation of already remotely used addresses. Both the receiver/transmitters and the addresses are mapped to the underlying protocols. In order to allow for the higher levels a protocol independent handling of resources allocated via the manager these have to provide a common interface. For the internal handling in the manager a similar isolation from the underlying protocols is required. This results in an isolation of the receiver/transmitter and addresses by an interface to the higher level and the protocols available, as shown in the above figure.

3.2.5 Design evaluation

While the above described design allows a very high degree of flexibility by using a modular approach and thereby providing a type of plug and play adaptability to any specific network environment, some improvements are possible. Currently the design limits the functionality to a common minimal denominator necessary for group communication. But with the developments of new network technologies and the increasing availability of networks including advanced functionality surpassing the pure data transmission (e.g., Active Networks as motivated in the introduction) a way is needed to integrate this functionality into CSCW systems. To achieve this a higher level of functionality is necessary than here presented, exceeding the limited ability provided by passing some initial setup options to the network service level. This is especially relevant if the system has to react to changes in the network. An example for this would be a change in the provided quality of service. This could be caused by a network problem or maybe by a price change of the used service (if pricing is, for example, done on a demand basis). In this case it would be necessary to inform the CSCW system of the change giving it the possibility to adapt, by, for example, reducing video quality. Events describing these dynamic changes will be asynchronous in nature and therefore require an extension allowing their propagation to the control level. Further it is possible that some of these events can not be handled automatically by the control level, thus, making a continuation of propagation on to the display level necessary. Here they are then

resolved via a user dialog. The interface to these intelligent networks is currently still uncertain, in the same way as their functionality. Here the ongoing developments have to be monitored and in time integrated into the network services level.

3.3 Tool integration in MACS

The tool level contains the application modules. These modules provide the functionality which is in the context of the whole system provided to the user as a tool to solve a specific problem. This section discusses general consideration for the integration of application modules into the framework, their interaction with it and the resulting design issues. Chapter 4 discusses specific examples of application modules.

3.3.1 Requirements for integration

The integration of tools into a CSCW framework is a difficult challenge. Each tool has very different requirements, not only with respect to resources, such as bandwidth (e.g., video), processing power (e.g., CAD) or memory requirements (e.g., slides in a whiteboard), but as well concerning advanced services required from the framework (e.g., floor control).

To achieve a flexible and modular design a set of programming interfaces concerning the integration of the application module are required. Therefore it is a suitable approach to define an interface through which other components, mainly the control, can access the application module, as well as a set of interfaces describing the services of the control. These interfaces will allow the framework to handle application modules in a manner independent of their media content and, therefore, allow the easy integration of new or improved application modules into the framework. As a summary the following requirements result:

- provide generic application module management
- allow easy integration of new/improved application modules
- provide common services to the application modules

3.3.2 Design goals for tool integration in MACS

These requirements lead to a number of design goals. In order to provide a generic management of applications it must be independent of their specific content. Therefore the application should only be accessed through a well defined interface. This combined with the implementation of the application as a module will allow an easy integration of new or improved applications into the framework. This is especially true if the application modules are loaded on demand and not rigidly build into the system (e.g., by hard coding certain application names). To achieve this the configuration must not be defined in the implementation code, but better in a separate (preferably textual) configuration file. The design of MACS uses such a configuration file. It is actually divided into two files, one containing the installation specific information normally entered by the system administrator or programmer setting up the system, while the other contains the user preferences. This allows a high degree of flexibility for adapting the system to any specific environment and user preferences without the need to recompile any code. The same mechanism as used for the integration of the application modules is used at any place within the framework which requires a flexible setup or scenario/user specific adaptations (e.g., the previously described support for different network modules dependent on the available infrastructure).

A further important point in the design of MACS was to move commonly used functionality out of the applications and into the control. This has a number of advantages based on the reuse of existing code. Application modules are more easily designed and programmed, as existing services offered by the control can be used and, thus, need not be designed, implemented and tested again.

As the result the following design goals can be identified:

- encapsulate application into a module
- use a well defined interface to access the application module
- file based configuration of application modules and their integration into the framework
- application modules should use services offered by the control if possible

The control services and the configuration file are discussed in more detail in the control section (3.4).

3.3.3 Scalability aspects

Scalability aspects will mainly depend on the application module itself, as well as on the network services used by the corresponding module. Some of the services offered by the system will be sensitive to scalability issues, but these are discussed later. The only remaining general point is the scalability of the application management, these are considerations regarding the number of applications which can be integrated into the framework without causing a scalability problem.

The application management is dynamic and does not limit the number of applications. System resources will place an upper limit on the number of applications, but these limits are mainly caused by available memory or disc space. Normally the limitations caused by the application modules themselves will be more severe (e.g., required processing power or the in section 2.4.2.3 and 3.5.2 discussed visualization aspects).

3.3.4 Structure of tool level

These design goals will result in a structure where each application is encapsulated into its own module. These modules can be accessed in a uniform way via the application interface, as, for example, done by the control. The application on the other side can access the control via a (number of) well defined interface(s). The control extracts the information required to manage the application module from the configuration file(s). The application module may optionally extract further specific configuration information from the same source. This is an indirect access. If, for example, the chat application module wants to extract the user configuration for the number of lines it should use to display chat messages, it will read this setting from configuration file via the control. To enable this the control merges during initialization the system and user specific entries into a unified representation. The merged version of the configuration is made available to the application modules and can be used as described above.

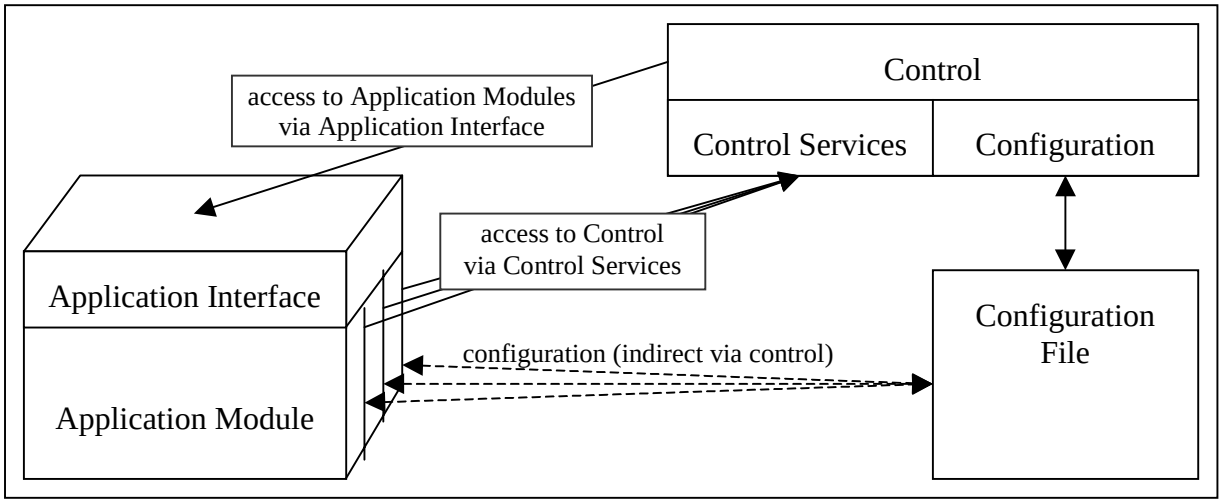


Figure 15: Tool level structure

While in Figure 15 only some examples of application modules are shown any number of these may exist, limited only by available system resources. Each application module has the same status towards the rest of the system, meaning that all applications are treated exactly the same by the system. This is necessary as no assumptions are made about the content of the application modules. This is done in order to guarantee a high degree of flexibility. An application module internal substructure will generally exist, but this is module specific and therefore not discussed here.

3.3.5 Design evaluation of tool integration in MACS

The design is again optimized with respect to flexibility allowing the integration of a wide variety of application modules while providing powerful support services via the framework. The integration of the applications in form of modules on the other hand imposes some small limitations. One is, that existing (non-MACS) tools can not easily be extended to fit into this system. Further, the decomposition of an instance of MACS onto multiple machines is not possible, as would be possible if the tools were standalone applications (i.e., running separated from the main system and not as a module within it). Such approaches, as, for example, MBUS [79] connect these single applications via the network using a specific protocol into one (virtual) system. The result is that a load balancing becomes possible if a number of prerequisites are fulfilled by the operating system. Considering an Unix based system it is, for example, possible using the X window system [39] to run an application on one host but to display the user controls on a different one. This would allow to select for each application the most appropriate host and to group these applications (each on a different host) together into a single instance of a CSCW system. As an example one could consider a setup with two video cameras, one for the speaker and the other one showing an experimental setup. Each camera is connected to a different machine. The experimental setup is controlled by a further machine. In this way the processing and memory requirements caused by using two video cameras (and therefore two video applications), the control application for the experimental setup and the control instance of the CSCW system are distributed across these three machines and not concentrated on a single one. Disadvantages of this approach are besides the increased complexity a number of security and access issues, due to the distribution of parts of the CSCW instance across multiple machines. As this is only feasible in a rather small number of cases and does not represent a typical situation, it has only limited effect on the overall usability of a system. Such an approach would as well raise a number of

important portability issues, as operating specific mechanisms are generally used to export, for example, the display of an application to a different host.

Thus, the design of the application module integration into the MACS framework overall provides a high degree of flexibility and portability. It allows fast and efficient application module development due to the support services offered by the framework while at the same time only imposing a small number of limitation for some specialized setups.

3.4 Transaction based control

The control is responsible for managing the complete system. As such it is the most important and most complex part of the MACS framework. This section will analyze the functionality required and, thus, derive a number of design goals. Next various scalability aspects are discussed, followed the structure of the control level and the interactions of its various components. Finally the advantages and limitations of the chosen design are evaluated.

3.4.1 Requirements for the control level

The management of the complete MACS system includes many different aspects. One of these is that information about the entities in the system has to be managed in form of a database. These entities are mainly sessions, users and applications, but further entities such as floor objects, various resources and internal modules (e.g., interworking) may exist. A well defined interface to access this information is required. Besides the database for storing information about these entities the control must as well provide detailed management mechanisms to manage the entities in the system. It must as well provide interfaces to access these management functions. This is again mainly of relevance for session, user and application entities.

The control itself must be flexible in order to adapt with the rest of the system to specific deployment environments. The integration of modules responsible for such adaptations should be easily possible. Important is as well the overall performance of the control. Especially considering scalability aspects the ability of the control to efficiently process a large number of requests while maintaining data consistency becomes important. As a result the following requirements can be identified:

- database support
- detailed and efficient control mechanisms
- modular extensible and adaptable to deployment environment (and usage scenario)

3.4.2 Design goals

From these identified requirements a number of design goals can be directly extracted. The control must offer database functionality mainly for users, sessions and applications. Hence it is important to define interfaces allowing all components to access the information in these databases. At the same time the control must update the data transparently, that is synchronize the data with other MACS instances, primarily those involved in the same session. This is important for acting on these entities in the database. To achieve this a number of detailed control mechanisms have to be supplied. These are then used to execute certain actions on the objects in the database, that is, for example, to expel a user (as found in the database) from a session (as well found in the database). To efficiently carry out such actions it is necessary to

allow their concurrent execution. An action aimed at, for example, joining a session will, in general, require communication with other MACS instances and therefore exhibit a relative large duration. Each action can be seen as a **transaction**, that is a job submitted to the system, then processed by the system during a certain amount of time and finally concluded with a well defined result (for alternative definitions see [72]). The term transaction is mainly used in the content of database access, as, for example, typically used in booking systems, but has already be used in the context of CSCW systems before [11]. An example for a transaction in MACS would be the join (i.e., the transaction JoinSession and its remote counterpart the RecvSessionJoin). A complete list of transactions is given in appendix A.2.1. For the example of the join the following applies. If the user selects a session to join and then initiates this process (by pressing on the join button) a JoinSession transaction will be created and submitted to the transaction system. The GUI thread (the thread handling events from the GUI caused by, for example, pressing a button) will be able to directly handle new events after submitting the transaction. All further processing of the join is now done within the thread of the JoinSession transaction. This includes the following steps:

- extraction of contact information for the selected session from the database
- local resource checks ensuring availability of resources to join the session
- assembly of join request
- transmission of request to session server
- waiting for response from server
- processing of response
- on success
  - initiation of local session participation
  - setup of local session controller
  - setup of session tab
- on error
  - provide user with error information
  - cleanup

If only one transaction can be executed at any given time the overall throughput and responsiveness will be fairly small, thus, resulting in poor performance and poor scalability [72]. A parallel transaction system will allow a much higher throughput, but is in its nature much more complex, as data integrity will be more difficult to achieve. Thus, the transaction system has to analyze for each transaction if the newly submitted transaction might interfere with other running transactions. In such a case the transaction must be postponed until the required system resources become available. It is important to note at this point, that transactions, especially those requiring communication between MACS instances will spend a large amount of time waiting for responses to requests send to the remote MACS instance. This makes the parallel processing of transactions viable, as no processing resources are used by the waiting transaction. In this case an increase in throughput is even on a single processor system possible. Otherwise enabling parallel execution will clearly not result in speed improvements on a single processor system [72]. On the other hand multi-processor systems are becoming more and more popular. Further, if control functionality in MACS is distributed onto multiple hosts, thus forming a group of servers, it will actually be possible to do real parallel processing, as certain transactions will be able to run independently of others and

therefore can be executed on a single host of the group of servers without interfering with transactions running on other hosts in the group. Responsiveness on the other side will always be affected by parallel execution of multiple transactions. The issues of dependencies of transactions is discussed in detail in section 3.4.5.1. In the design of the transaction system it is important to decide on the granularity used to define transactions. A finer granularity will result in more efficient parallel processing, as long as management overhead is not considered. For real systems one has to consider this, thus a finer granularity will cause a greatly increased complexity of the dependencies, as well as an increased management overhead. This will oppose the positive effects and might as well surpass these, resulting actually in a degraded system performance in the case that the effects of management and dependency checking outstrip the improvements gained by parallel processing.

For the MACS transaction system the following types of setup were considered:

level	**type**	**parallel execution**	**exclusion (dependencies)**	**MACS**
1	system	only a single transaction in the whole system	complete exclusion	yes
2	session	only a single transaction per session	session level exclusion	yes
3	action	multiple transactions per session	transaction level exclusion	default
4	micro	transactions subdivided into worksteps	workstep level exclusion	no

Table 13: Transaction system levels vs. granulartiy

Level 1 is the simplest setup and basically emulates the sequential case by just allowing a single transaction to be executed at a time, thus enforcing a complete exclusion of other simultaneous transactions. This setup is supported by MACS and can be activated via a corresponding property file, but is basically only relevant for comparison purposes.

Level 2 allows the parallel execution of transactions belonging to different sessions. This setup is as well supported by MACS, but again not really of practical relevance, as scalability and performance issues are unlikely to arise due to the number of parallel sessions managed by the system (see section 2.4.1.2).

Level 3 allows in principal parallel execution of transactions in the same session, but defines some exclusions via a list (as per definition) assembled by the system/transaction designer. As a result a transaction can, if necessary, ensure that other transactions which could cause conflicts will not be allowed to run at the same time, while the normal case assumes parallel execution. As this is the minimum level necessary to efficiently handle expected system loads (see section 2.4.2.2, especially Figure 9) for larger sessions, such as the class room or lecture scenario, this level is the default setup for MACS.

Level 4 will allows most transactions to execute in parallel, as here a finer granularity is used to define exclusions. For this a transaction is subdivided into worksteps. The resolution of conflicts will now have to not only consider all possible combinations of transactions, but as well all possible combinations of worksteps for all transactions, thus, resulting in a drastic increase in complexity and a significantly increased management overhead. The finer

granularity will not only have an effect on the conflict resolution, but the complete system has to deal with a finer granularity. This setup is not supported by MACS. Here a detailed analysis would be necessary in order to determine the possible advantages of this setup versus its added complexity.

The structure of the transaction system and the details for the dependency resolution are given in section 3.4.5.1, while an evaluation (including measurements) is contained in section 6.5.

While the databases and the transaction system are the heart of the control some further issues regarding flexibility and extensibility have to be regarded. The control should at least be as extensible and adaptable as the rest of the system. Therefore special attention has to be paid to the modularization of the control. An important aspect is the integration of specialized modules, as, for example, to provide interworking capabilities. These type of modules will have to access the internal structures of the control and may even need an integration into some of the control mechanisms. Still these modules should only be optional. The control must therefore remain functional even if certain modules are not loaded. To achieve this type of integration a well defined module concept has to be established featuring clear interfaces. Further, the control must provide a number of hook-in points to allow the modules to integrate themselves deeply into the function of the system, if so required. It is important to stress the point that these modules by their design and aims are in no way related to the application modules introduced before.

The main design goals for the control, as relevant in the frame of this work are therefore as follows:

- well defined interface to databases
- control mechanisms for objects in databases
- efficient transaction system to allow scalability
- support for optional modules with hook-ins to control internal functions

Further design goals do exist for the aspects of control strategy (which is part of the overall control) and control visualization (which is part of the GUI level), but are not in the scope of this document (see [19]).

3.4.3 Scalability aspects for the control level

A number of scalability aspects exist for the control level. These cover quite different issues and depend on the deployment environment and usage scenario.

- size of database
- throughput/responsiveness
  - processing of database queries
  - processing power required for control mechanisms

The size of the database is mainly dictated by the number of entities in it. For the main entities being applications, sessions and user the following will apply:

The number of applications contained in the database will have a small effect. This can be seen by the comparison in Table 14, showing the time taken to retrieve entries from the database. The number of applications in the database will be a small number, limited by the available and installed applications and, thus, be likely smaller than 10. Even for large number of applications (as about 100) the times to access these entries is not critical. As database access is handled for all MACS databases by the same mechanisms this times will equally apply for access times to the session or user database entries. These databases can be substantially larger but as shown in Table 14 even for a database with 1000 entries the access time is quite small. Regarding not the time needed to access the information, but memory and processing requirements one will, in general, find that the application itself will consume a multiple of memory and processing power than their database entries. It is clear that a video application module will spend more time on coding and decoding then a database access will cost in terms of processing power, the same as that a whiteboard will generally require much more memory to manage its set of slides than required by its entry in the database.

number of entries	**1**	**10**	**100**	**1000**
retrieve entry [ms]	<0,10	0,10	0,11	0,16

Table 14: Database access times (not concurrent)

The number of sessions is more relevant. It is quite imaginable that the session list alone will have hundreds of entries and that these will be updated or modified by a large number of messages received by the local MACS instance and passed on to the control for processing. Nevertheless, this will again be much less an issue than the actual number of sessions the local system is joined to. Here the times from Table 14 for the database access apply again. The time to update some of the fields of such a entry once retrieved from the database will in generally be very small and consist mainly of assignments. By far the most intensive operation in terms of processing power is the decoding of the incoming update messages and their submission to the transaction system. The time taken to process these transactions is clearly dependent on the total load on the system, which will equally apply for each transaction submitted within a joined session. Thus, the number of joined sessions will cause a much higher load on the system, not only as it is in the same way affected by an increase in system load, as is the database, but it as well has to handle a larger number of operations. A session update will only be generated due to a change in the session. This change will have to be brought about by some local or remote user interaction. In such a case the mechanisms to handle the user request are initiated, normally resulting in a number of transactions. Once these are processed the session data is updated in the session and then a global update is send by the master. Thus, for each global update received a much more time and processing intensive process will have been executed in the corresponding session. Fortunately the number of simultaneously active sessions is limited indirectly by the user. The user, in general, will not be able to simultaneously follow many different sessions, or even to participate in each of them. This leaves the number of users as the main factor determining scalability. This factor will be the most relevant, as there are typically many more users than sessions. To keep the user information updated is a challenging issue and might put a substantial load on the system (if one considers to keep the information about all possible users of the system worldwide up to date a very large number of many millions of potential users would arise). Fortunately again most of these users are not directly relevant to the local user. His ability to manage or to view lists with thousands of users is rather limited. Therefore a more relaxed schema for updating the user information (not immediately, for example, but

only every *x* minutes) will be acceptable. Further, the user will normally limit the group of people he is interested in by himself. This can be done, for example, by placing a geographical restriction on the list of displayed users. As a result the user entries in the database and the related overhead in processing corresponding messages and requests will probably be comparable to the overhead caused by the sessions.

Overall it can be concluded, that while the database will consume some memory and require some processing power, this will typically not be the factor limiting scalability, as access times are quite small, as shown in Table 14, and internal updates are mainly limited to straight assignments. Despite this it might be a good approach to integrate mechanisms automatically monitoring the database regarding its size and resort or reduce it according to the statistics created during monitoring. This would assure that even if the user were trying to follow **all** data about **all** sessions and users, the system would remain functional. The result would be a time penalty for accessing data not found in the local database as it will be necessary to extract it from the network on request.

The throughput when processing requests is a very important factor. Here the limiting factor is not necessarily the processing power available, but the efficient usage of it and hence the responsiveness of the system as perceived by the user. A better performance can here be achieved by allowing the parallel processing of multiple transactions. Important is hereby to guarantee the integrity of the databases. This is not simply done by providing locking mechanism, as a transaction might be based on data retrieved at the beginning of the transaction, which could have changed by the time the transaction finishes. Therefore locking the database only during the process of retrieving or updating the information will not guarantee data integrity, as an other request might modify the data in the meantime (see Figure 16). Locking the database for the whole period of the transaction on the other hand will provide data integrity but does not allow parallel processing of transactions and, thus, will lead to performance problems (see Figure 17). If a transaction system is used conflicting transactions are postponed and only transactions with no critical interdependencies are executed at the same time. This combined with locking on a resource level basis (see Figure 18) is a much more efficient method, as can bee seen by comparing Figure 16 to Figure 18.

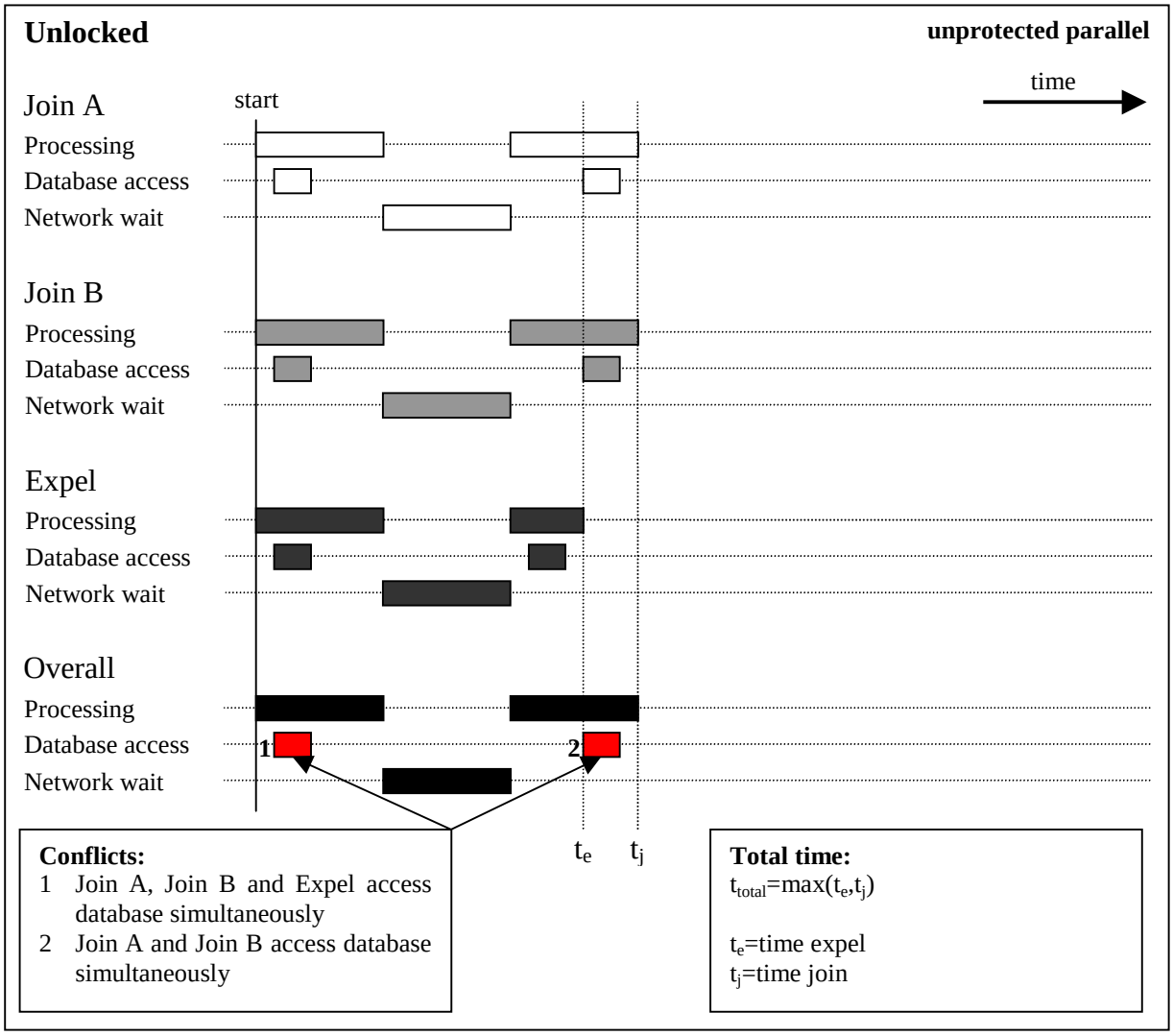


Figure 16: Example of unlocked database access

As shown in Figure 16 conflicts can arise at a number of points. The main problem are the conflicts caused due to changes between database access by the same transaction (e.g., Expel) as indicated by the second conflict. Conflicts based on the simultaneous access can be fairly easily avoided by locking the database for the duration of the access, which does not help to avoid the other conflicts. The time for the system to execute is given by the maximum time taken by any single operation and, thus, the fastest of the three possibilities, but correct results are not guaranteed.

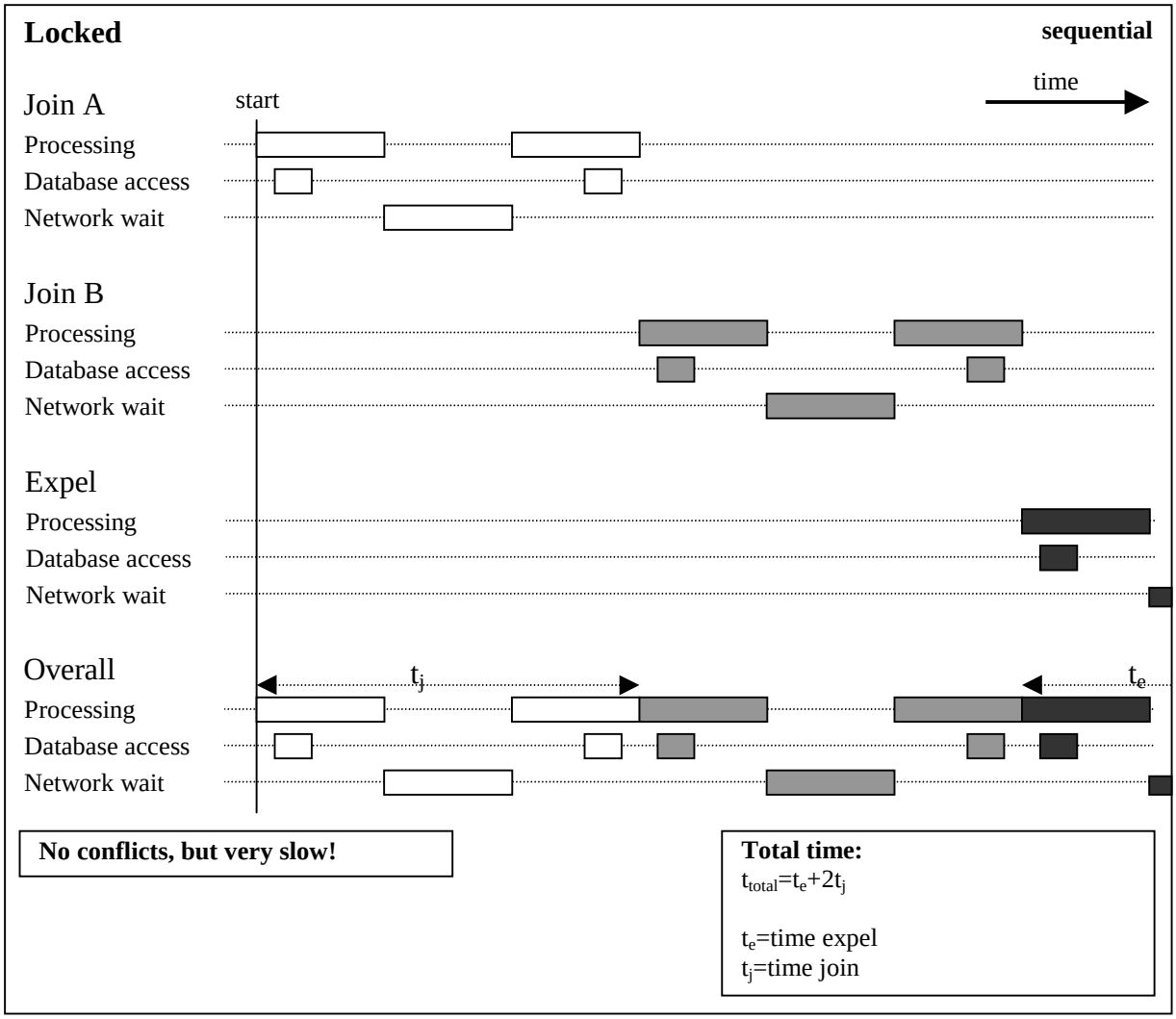


Figure 17: Example of simple locking for database access

The Figure 17 shows a simple but very inefficient way to avoid conflicts due to overlapping transactions, by simply executing the transaction sequentially. While in some cases there is no possibility to prevent conflicts but to ensure that certain transactions are not executed in parallel this is generally not the case. As shown by the overall chart this approach is not an efficient usage of available resources. The total time will here be the sum of all single transaction execution times. This is clearly much larger than the maximum of the single transaction times, but at least the results will be correct.

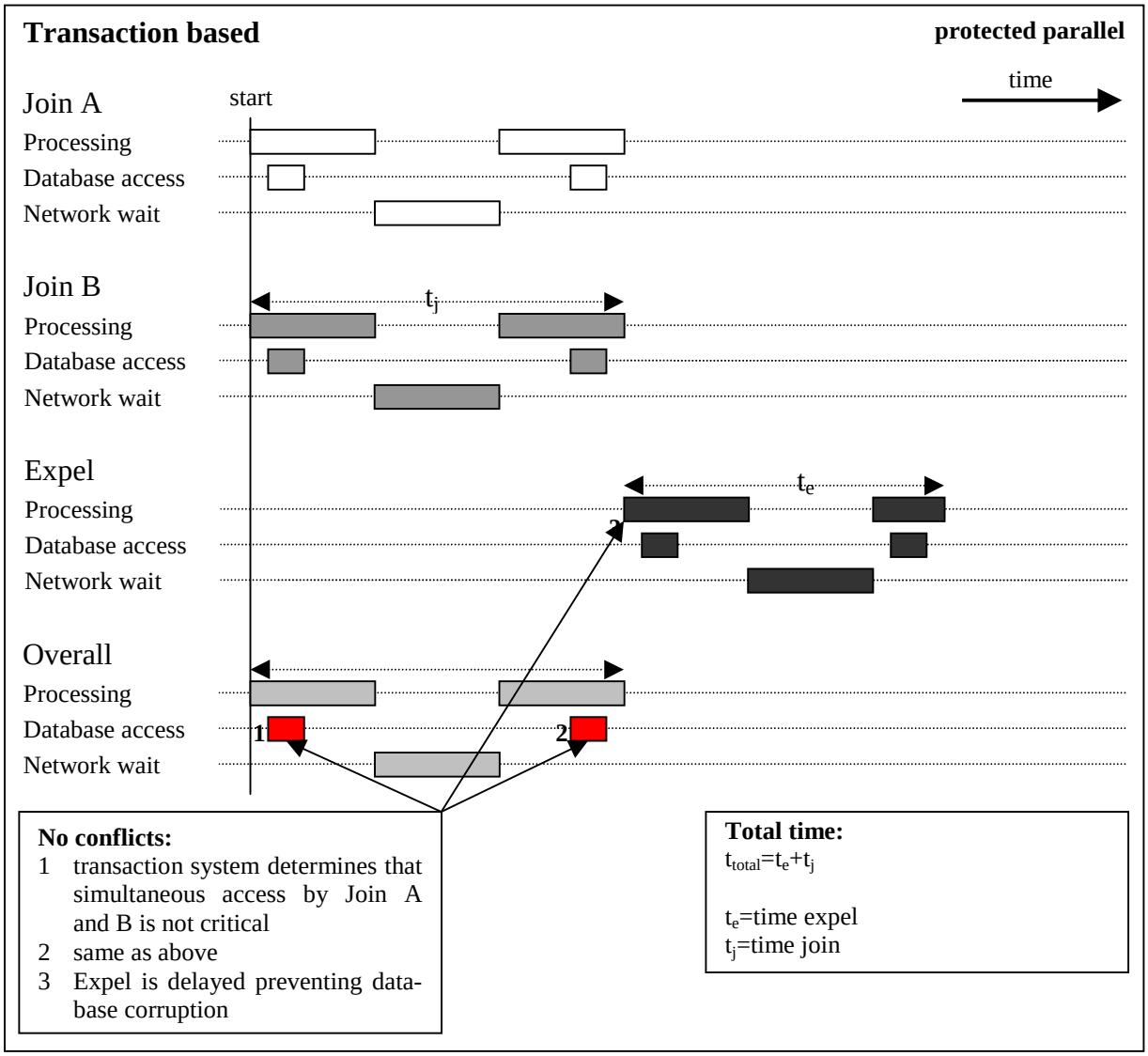


Figure 18: Example of transaction based database access

In the Figure 18 the transaction system prevents parallel execution of conflicting transactions and allows all others to execute in parallel. In this way data integrity is guaranteed, while the performance is clearly better than the sequential locking approach. This is shown in Table 15 for a direct comparison for the example illustrated by Figure 16, Figure 17 and Figure 18. Further, more than the here shown transactions could be executed simultaneously, as, for example, the time A and B wait for a response from the network is not used by any process.

	unprotected parallel	**sequential**	**transaction system**
total execution time	$\max(t_j,t_e)$	$2t_j+t_e$	t_j+t_e

Table 15: Execution time comparison (2 joins and 1 expel)

Looking at scalability not from the component view but from a scenario view a couple of further points can be noted. For small scenarios, as the discussion group, no problems are expected. The class room scenario will normally only stress the control during the initial setup

phase. In that phase the participants of the session will join the session all within a typical short period of time. This is equivalent to the students all entering the class room after a break. As the control is the virtual door regulating the access to the class room, that is it regulates who is allowed in, the initial phase will typically show many more requests per time unit than during the normal execution of the session. The most interesting scenario is the podium discussion. Here the number of participants can be very large. Further the participants will typically form a much more dynamic group, with a more or less continuos amount of people entering and leaving the session throughout its existence. Again an initial phase with a very high load on the control will exist where the majority of the participants enter the session. Here scalability problems are expected. Two approaches to reduce these problems are viable.

First an increase of the available processing power and thus as well throughput is possible. This can be done similar to the real world by providing a larger number of access doors instead of a single one. For the computer supported variant this would be to provide a number of controls able to process the join requests. A group of MACS instances working together will be able to manage an increased number of transactions but on the other hand will have to deal with increased synchronization overheads. The integration of such a distributed access mechanism will have an effect on various components and is not just limited to the control. This approach realized in MACS by providing support for a group of server setup [42].

The second approach is to reduce the cost of (i.e., the time and resource required to process) the relevant transactions. This can be achieved by a hierarchical setup. In this setup plain passive participants (i.e., listeners) are treated differently. They are not active and will therefore neither cause messages nor provide data needed to mange the session. Exploiting this, a hierarchy for the type of participants can be created and thus a set of lightweight transactions can be defined. The case of a hierarchy with only two levels should only be seen as a starting point. More complex hierarchies based on the different roles of the participants are possible, as, for example, discussed in section 2.2. Only if such a passive participant becomes active (by, for example, raising a question) he will be shifted into the active group of participants, using the current (full) set of transactions. While this is mainly a concept important to the control level a direct support by other levels would additionally increase its effectiveness, as further discussed in section 3.6.

3.4.4 Structure of the control level

The control level is structured according to its required functionality into the following components, as shown in Figure 19:

- Databases
- Enhancements (internal modules)
- Communication services
- Transaction system
- Session controller
- Floor control

Each of these components will be described in more detail in the following subsections, except the latter two components which are not in the scope of this work and are explained in detail in [19].

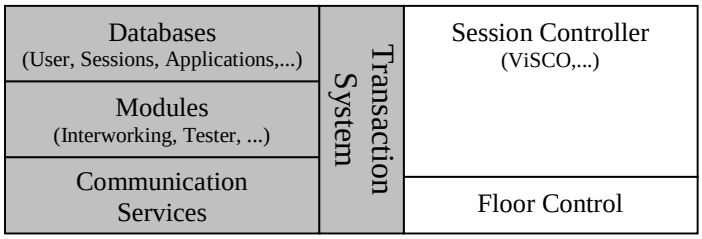


Figure 19: Control level structure

3.4.4.1 Control databases

The access to the databases allows the extraction of user, application and session data. The user database contains information about all users know to MACS. The application database stores information about the locally installed applications and the applications currently used in the known sessions. These two are different, as in a session a participant might use an application which is not available locally. The session database provides access to information about all currently known sessions. Each of these databases can be accessed via the control allowing any application or module to extract the required information about users, sessions and applications.

The databases storing information about applications, users and sessions are interconnected in a number of ways. For example a session entry contains a list of participating users. Figure 20 shows these relations. For each of the three basic entity types a manager exists. These offer typical functionality for manipulating the database. Each of the types is stored in a Hash table, generally offering fast access to the entries via a hash key. With growing size of the database the usual performance characteristics of hashing apply. While for a large database access times will increase, they are not a limiting factor in the overall processing of requests. This can bee seen by comparing database access times as given in Table 14 with transaction times as discussed in section 6.5. Here one can see that while the database access time for a database size of a single user is below 0,1ms and grows to about 0,16ms for thousand users in the database, the execution time for transactions will be in the order of 10^{th} of ms for a single transaction executing at the same time to 1000^{nds} of ms for hundreds of simultaneous transactions. Thus the access time is not the problem. The lockout time on the database on the other hand are much more problematic. This is the time the database can not be accessed because it is locked due to another process acting on it (see section 3.3.3). This did lead to the design of the transaction system in order to optimize processing of requests, especially considering efficient database usage while maintaining data integrity.

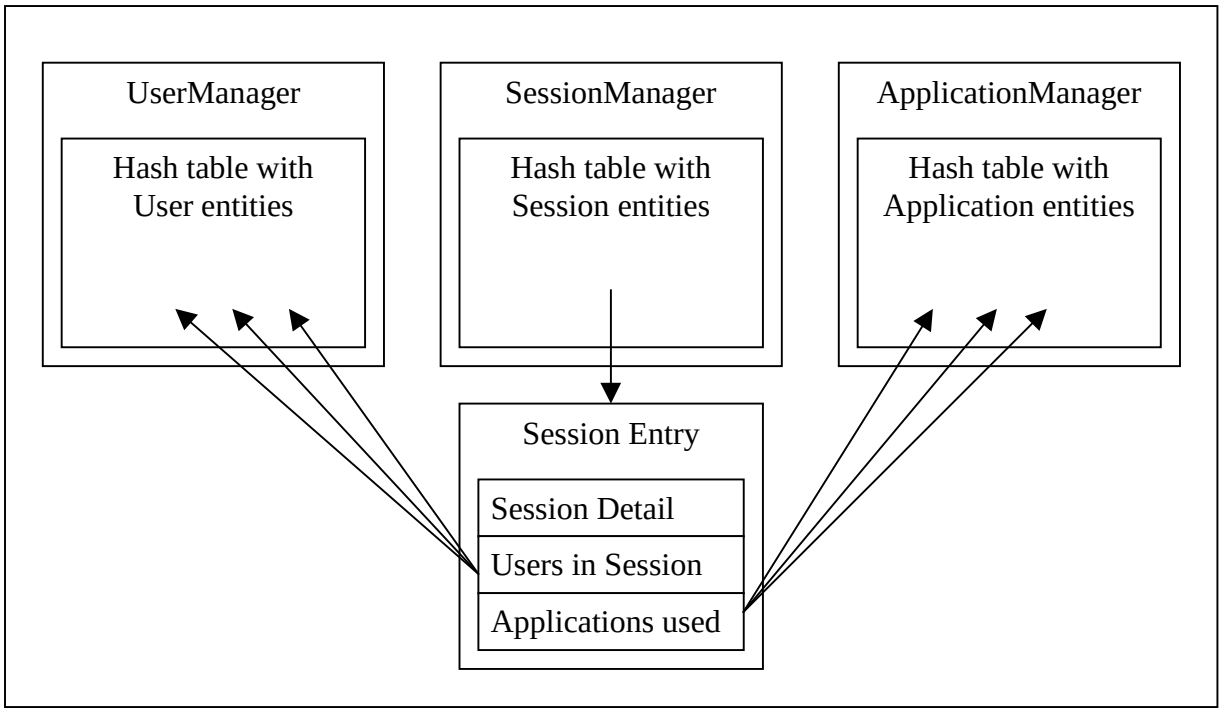


Figure 20: Database internal structure

The user entries contain the details about the user. These are typically his name, contact address (including email and web addresses), as well as his telephone number. Further system relevant information are stored. This includes the system the user is logged on, his logon name and his detected hardware. The hardware found is relevant to enable certain applications which depend on the detected hardware, as, for example, it is only useful to offer an audio sender if a microphone and audio capture system is present.

The session entry is much more complex. It contains detailed information about the session, including the session name, its creator, a short description about the content of the session and the session type. Session type refers here to the way the session will be conducted, for example, as lecture or as discussion group. This choice will result in the automatic selection of a suitable session controller, such as, for example, ViSCO [19]. Further, a list of participants (users) taking part in the sessions is kept. These are references to the entries in the user section of the database. Finally, a list of applications (actually together with whom is using them) is kept. Here the main reference is towards the corresponding entries in the application section of the database. In this case not only a reference to the type of application is used, but as well a local instance has to be created and is managed in this application list. This is necessary, as an application class can have many instances in different sessions, which normally are considered independent entities. Thus the application entry managed by the *ApplicationManager* is the actual class loaded by the configuration system, as explained in section 3.3.4.

An example for each of the entries, that is for a user, session and application entry respectively, is given in appendix A.1.

3.4.4.2 Enhancement modules for the MACS control

Modules are one of the MACS system extension mechanisms. A module is loaded at startup (but could as well be loaded while running) according to the configuration file. The configuration file describes the local configuration of the system and can be user specific. It

contains setup information for the local sessions, determining, for example, what type of network will be used by the system. Each module will offer a specific service/functionality. These can be very different in nature. Examples included with the MACS system are the interworking modules (MBone and ITU/ILS) and test modules (Tester and Viewer). These modules offer a wide variety of different services.

The MBone module allows access to MBone based systems. It enters announced MBone sessions into the session list and announces MACS sessions on the MBone, thus allowing MACS and MBone participants to join the same session (if compatible tools are installed). The ITU and ILS (Internet Location Server[1]) modules are similar. The ILS module [64] enters the ILS users, e.g. NetMeeting users, into the MACS user list, as well as the MACS user into the ILS list. NetMeeting users can now be invited into MACS sessions or contact MACS users. The actual communication is done via the ITU module. The interworking modules are discussed in detail in chapter 5.

The test modules on the other hand serve a very different purpose. They allow automatic stress testing of some MACS functions. They can log performance data during these tests and allow insight into the database content at any given operation time. A wide range of other modules is possible depending on the additional wanted functionality. Modules in MACS have a similar role as plug-ins in browsers.

3.4.4.3 Communication services of the control level

The communication services simplify the communication among the control level of MACS instances (system messages), as well as the communication between session controllers and floor controls (session messages). System messages serve for announcing system information and their changes (e.g. data about the user of the local system). Session messages are used to manage administration and synchronization within a session including the involved floor controls. These messages are, for example, used to create, join, leave or terminate a session. Further, if a group of servers [42] is activated the communication services will provide the synchronization support and the channeling of server specific messages (e.g., messages were the person being the session chair has to react on).

Thus, the communication services are responsible for parsing data received from the network and to initiate corresponding transactions with the local transaction system, as well as to pack and transmit data in form of requests, responses or indications as generated via the transaction system. To achieve this the communication services have two basic types of building blocks, the *SystemNet* and the *SessionNet*, as shown in Figure 21, as well as an optional *ServerNet* building block only relevant for setups using groups of servers.

[1] A type of X.500 server defined by Microsoft to serve tools as NetMeeting to exchange user information.

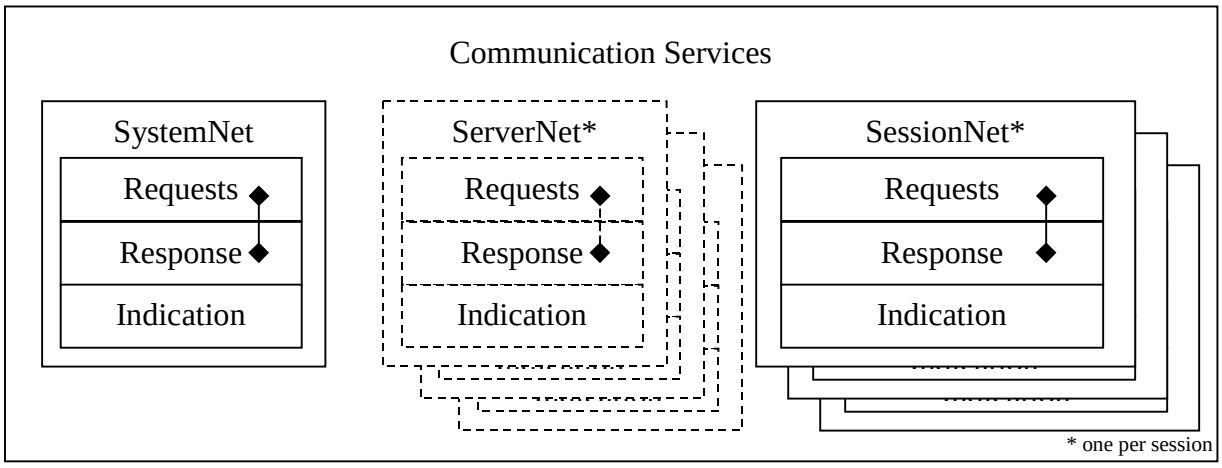


Figure 21: Internal structure of communication services

Only one copy of the *SystemNet* exist, as it handles all messages which are directly related to the local instance and therefore are session independent. On the other hand, a *SessionNet* for each session exists handling all session relevant messages. For each session using a group of server setup a *ServerNet* does exist[2]. Three types of messages can be generated, a request, a response and an indication. The request is a question direct towards another system, for which a response is expected. This means the requesting transaction will wait until such a response arrives, or until a previously set timeout occurs. This will happen if the other system cannot or does not want to respond. The indication on the other hand has a more informative character. No reply on an indication is expected.

In general complex operations as, for example, join or invites will result in the exchange of a request/response pair between the two involved instances. If this transaction concludes successful generally an indication will be generated to informing the others not directly involved in the transaction that, for example, a new participant has joined the session. Less complicated operations such as a leave require only a single indication (see section 3.4.5.2).

3.4.5 A functional view on the control

Having discussed the structure of the control level in more detail and marked some interesting points in the previous sections, it is still necessary to have a closer look at the interaction of the various components. Figure 22 shows a detailed view of the MACS system with special focus on the control level details. Only a single generic application module is shown. The arrows indicate if access between components is uni- or bi-directional.

Within the control all interactions are done via the transaction system, efficiently isolating the control components. Only three possibilities exist to access these control components from the outside. First, a direct access to the transaction system is possible in order to submit a transaction (Figure 22, arrow A). This is mainly used by internal modules. The second possibility is through the communications services (Figure 22, arrow B). This is used by the network services. Each incoming message is analyzed by the communication services and then entered in form of the corresponding transaction into the transaction system. The inverse equally applies, that is, each action within the control resulting in a message created by the

[2] The group of server approach, as remarked before, aims at distributing the management functionality for a session onto a group of hosts acting thus as servers. This approach is currently being integrated into MACS.

communication services and send via the network services will pass through the transaction system. The third way of access is through the user interface (Figure 22, arrow C). This can happen by just passing the information directly from the application panel through the control to the application module, or from the controller panel to the session controller, who in turn might submit a transaction to further process the user selected action. The various components of the network services are managed by its manager (as shown by the arrows connected to the outside of the network services box containing the manager), but are as well directly accessible once allocated by the manager (as shown by the arrows directly connected to the resources or the receivers/transmitters). The communication services, applications and some of the internal modules will use the transmitter/receivers to distribute their data to other MACS instances. The corresponding addresses are entered into the databases and are directly used by the applications.

The following subsections will describe the design of the transaction system and the protocols used the by communication services in more detail.

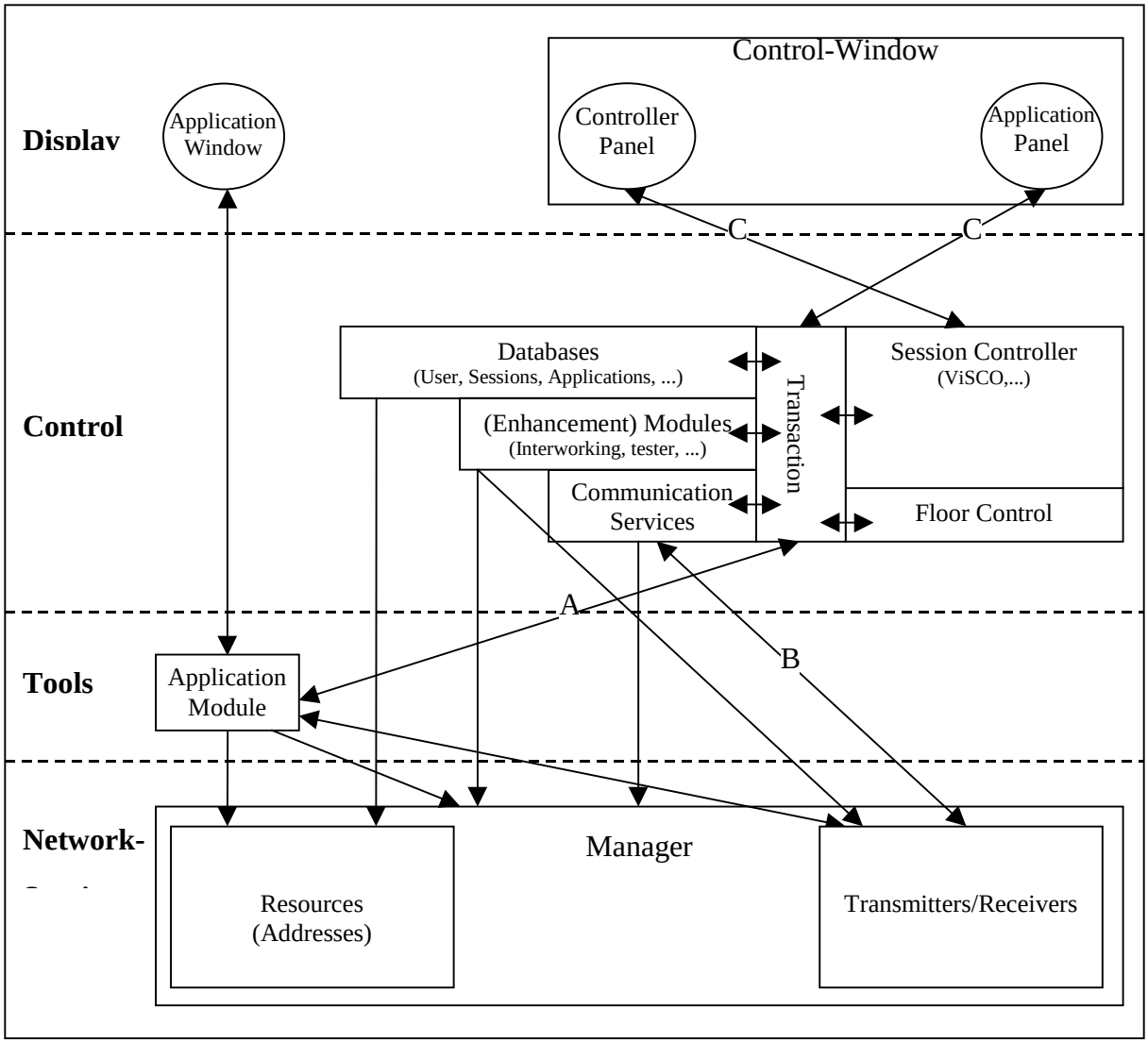


Figure 22: Interactions of the framework components with the control

3.4.5.1 The transaction system

The transaction system is a key component of the control level. It is responsible for scheduling the different operation the control has to fulfill. This is done in form of transactions in such a way as to guarantee data integrity while optimizing throughput (as motivated and described in section 3.4.2). The rest of this section describes the internal design of the transaction system, configuration relevant aspects including dependency resolution and issues concerning the integration into the control.

The internal structure of the transaction system can be subdivided into four basic building blocks:

- the input queuer (FIFO)
- the dependency checker
- the set of running transactions
- the set of waiting transactions

To initiate an operation to be executed by MACS the corresponding transaction object is created. It is then submitted to the input queuer, as shown in Figure 24. From here the Transaction is delivered to the dependency checker. Two types of checks have to be performed at this stage:

- sequential dependency check
- parallel independency check

The sequential dependency check is needed to ensure that actions will only take place in a certain order [2]. Figure 23 shows these sequential dependencies for user initiated session control transactions. An example is that a session might only be terminated once it is created, or that a user may only be expelled if he has in some form (join or invite) previously entered the session. These dependencies can be subdivided into four basic types, as shown in Figure 23. These is the main interactive sequence of the system allowing to create a session, join a session or exit the system. If a session is created the interactive sequence of the session:master (that is the session as seen by the session chair) is enabled. Here it is possible to invite a user or to terminate the session. If a user is invited the user:master (that is the user as seen by the session chair) sequence becomes available, basically allowing to expel the user again. If a session was not created but an existing session was joined from the system sequence, the interactive session:slave (that is the session as seen by a non-chair participant) sequence becomes available. This sequence only allows to leave the session again.

These sequences are limited to basic functionality. Additional functionality especially in cooperation with the session controller are available (see as well [19]). Further, logical restriction based on current database state are not shown. It is, for example, not possible to create two sessions with the same name.

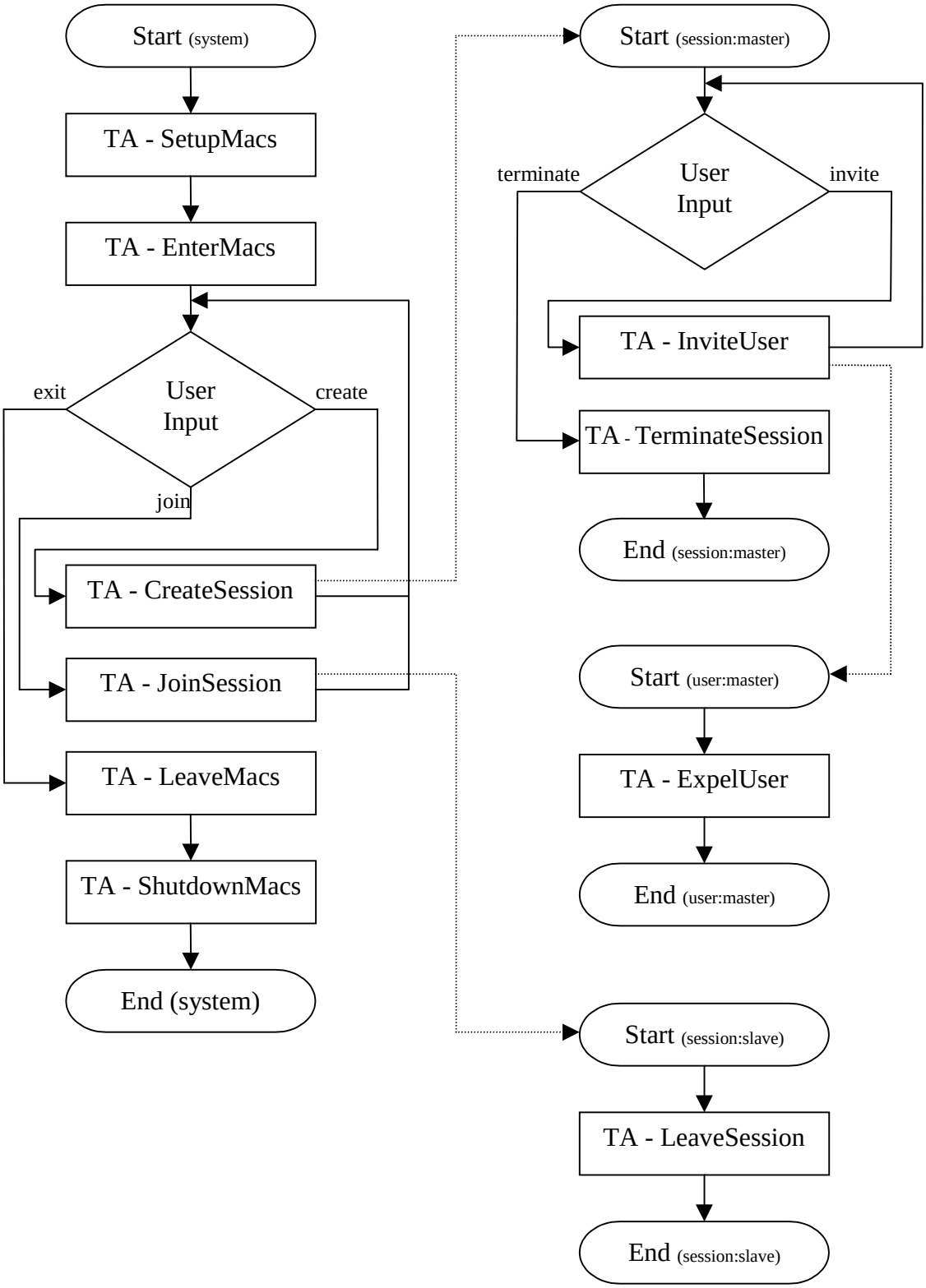


Figure 23: Dependency graph of transactions (TAs)

The other point is the parallel independency check, that is the check to ensure that a transaction can run independently of the currently running ones and thus no conflicts will occur during its execution. The MACS transaction system uses a granularity which considers a complete transaction at a time, as motivated in section 3.4.2. After the transaction has been submitted, it will make its way through the input queuer (FIFO). Then the dependency checker will determine if the transaction can be placed in the running set. This will happen if

no conflicts (the resolution is described later) exist, else the transaction has to be moved to the waiting set. In case of it being placed in the running set the transaction will start its execution. Once execution completes it will be removed from the system (trashed) and hereby trigger a reevaluation of the waiting set. This is done by transferring all eligible transactions into the input queuer. Some transactions, mainly periodic ones, might not be eligible, as these were after their execution not trashed, but put to sleep (in the waiting set) for resubmission after wakeup. In case of a possible conflict detected by the dependency checker the transaction is moved to the waiting set were it will remain until a reevaluation, as just described, occurs [66].

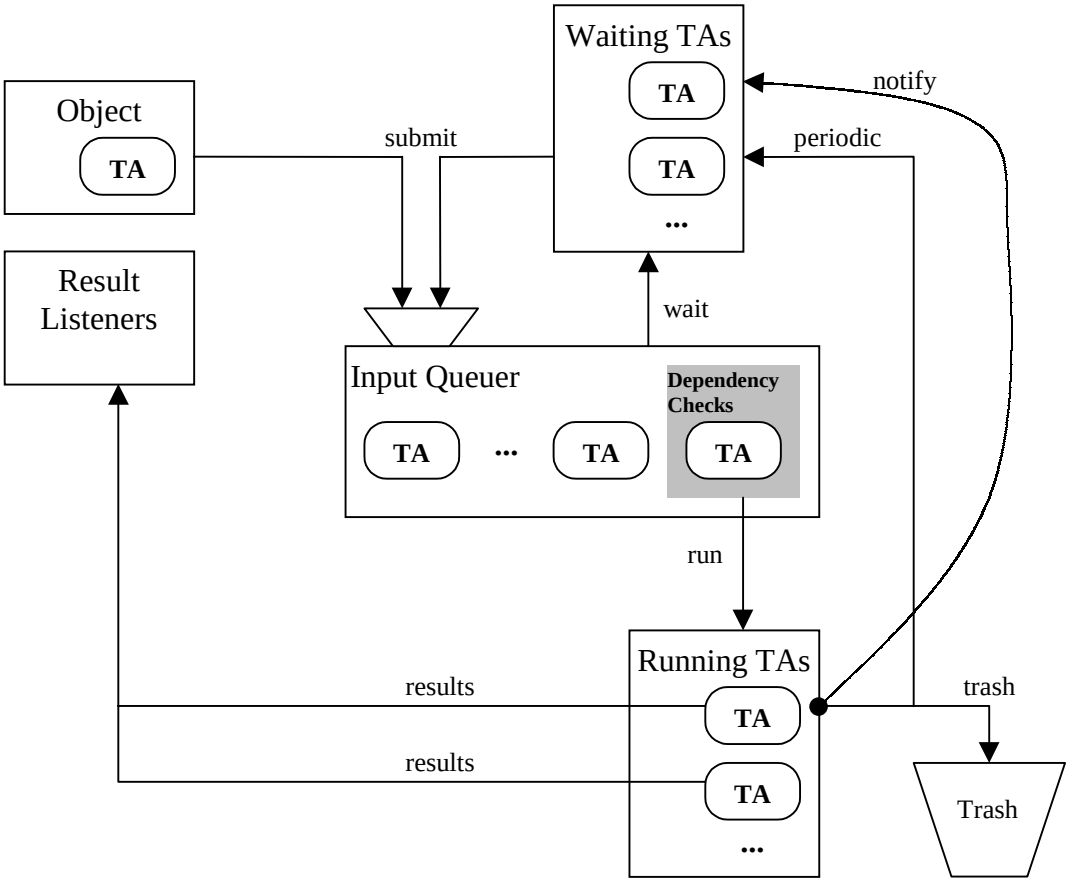


Figure 24: Structure of transaction system

Especially considering the initiation of transactions via the GUI, which is a common case (e.g., selecting a session and pressing the join button will initiate a join transaction) de-coupling becomes an important issue. This is achieved by two measures. First of all each transaction is a thread in itself. Secondly the submission of a transaction to the transaction system is asynchronous. If a monitoring of the state of the transaction is wanted this can be done via a supplied result listener. This allows, for example, to provide a user feedback via the GUI.

The structure of a transaction itself is mainly determined by the in Table 16 given fields contained in each transaction (further fields with implementation details exist, but are for functional considerations unimportant). The name/type of a transaction is used as a basic reference. This could, for example, be *JoinSession*. The name/type is later on used by the dependency checker. The group establishes a further parameter for the dependency checks. The default defines the **system** and **session** group, but it is possible to define any number of

custom groups as needed (e.g., by extensions defined by future developments, as well as by internal modules using the services of the transaction system). A key element is the independency field, which is used by the dependency checker in order to determine if a transaction should be moved from the input queuer to the running or waiting set. If a transaction is not allowed to run with any other, i.e., it can only run while no other transactions are running, its independency has to be defined as **none**. The independency for a transaction which is not allowed to run simultaneously with any other in its group (in MACS the group normally corresponds to a session) must be set to **groupnone.** Finally, if a transaction is allowed to run with others of its group (or as well outside its group), then an exclusion list can be provided specifying which other transactions are not allowed to run simultaneously. The state of the transaction indicates its current processing status. Once a transaction is submitted to the system its state is set to submitted (SUBMIT). This will remain unchanged until the transaction is moved to the running set, changing its state to started (START). Once the transaction has finally been concluded it will be removed from the system. A number of objects registered as listeners might exist. This is especially useful for de-coupling and feedback purposes.

field	**values**	**description**
name/type	see appendix A.2.1 for a complete list	identifies the type of the transaction by a given name
group	• system • session • custom	allow to define group specific behavior/ dependencies
independencies	• none • groupnone • list	define independencies
state	• submitted • started • done	indicate current state
listeners	list of objects	list of objects informed about transaction status

Table 16: Basic structure of a transaction

Each transaction is not only a thread but as well adheres to a strict interface. As a result it is easily possible to integrate new transactions into the system by defining a corresponding implementation in compliance with this interface. This is very important in a modular system in order to maintain flexibility and to allow the easy extension of the control mechanisms (e.g., by implementing new functionality). To achieve this a dynamic definition of the independencies between the transactions is required as these can not be hard-coded for a flexible systems such as MACS.

Before having a look at the algorithm used to resolve independencies it is useful to have a look at an extract of the default setup for the independencies in MACS, as shown Table 17.

per system		**per session (i.e., once for every active session)**	
locally initiated	**remotely initiated**	**locally initiated**	**remotely initiated**
	SystemUpdate		
	SystemExpire		
SystemAlive SystemUpdate		SessionCreate	
SystemAlive SystemUpdate		SessionTerminate	
SystemAlive SystemUpdate		SessionJoin	
SystemAlive SystemUpdate		SessionLeave	
SystemAlive SystemUpdate		SessionUpdate	
SystemAlive SystemUpdate		SessionInvite(s) SessionExpel(s)	SessionJoin(s) SessionLeave(s)
SystemAlive SystemUpdate			SessionCreate
SystemAlive SystemUpdate			SessionTerminate
SystemAlive SystemUpdate			SessionInvite
SystemAlive SystemUpdate			SessionExpel
SystemAlive SystemUpdate			SessionExpire

Table 17: Extract of default independencies (system and session group)

Table 17 shows the independencies, that is the lack off dependencies between transactions in MACS (how these independencies were derived is explained later). This is only an extract containing mainly the for this discussion interesting transactions. A complete list is given in appendix A.2.2. The table shows the two by MACS predefined cases, the system group and a representation of a session group. For each session a separate group will exist, thus sessions are independent of each other. As a result the total number of groups is one more (i.e., the system group) then the number of active sessions. All session groups posses the same structure therefore it is sufficient to consider one as a representative for all of these. Further a division into locally and remotely initiated transactions is done. Locally initiated transactions are normally initiated by the user via the GUI (e.g., by selecting a session and requesting to

join it) or via periodic transactions (those repeating themselves in a predetermined time interval) started automatically during system initialization. Remotely initiated transactions are initiated by requests received via the network from another system/user. Each row of the table contains transactions which are allowed to run at the same time. The most restrictive transactions are the remotely initiated *SystemUpdate* and the periodic *SystemExpire*, as these do not allow the parallel execution of any other transactions. This is achieved by setting their independency to **none**, as explained in Table 16. Next, transactions such as *SystemAlive* and *SessionCreate* are only allowed to run if no other transactions in the same group are currently running. As these two are in different groups they actually can be executed in parallel. Thus one can conclude that it is possible to create multiple sessions simultaneously. While this is true, this will, in general, not be really a common or greatly useful case, as the number of simultaneously active sessions is not really a scalability issue (see section 2.4.1.2). *SessionCreate* is as well a good example of another fact, being that most session transactions are not really eligible to execute in parallel. As in this case, it is not possible by the nature of a session to simultaneously create the same session more than once, as already shown by the sequential dependencies in Figure 23. This is equally true for operations as join, leave or terminate. Therefore, them not being allowed to execute with other session operations simultaneously is not a limitation imposed by the transaction system design or implementation. The gray shaded row is the main row of interest, not only as it allows the most parallel execution of transactions, but it as well represents a critical situation in the sequential case (e.g., level 1 as defined in Table 13). These are the operations which can and are likely to be executed in parallel. While it makes no sense to try to join the same session multiple times in parallel it makes sense to process incoming join requests in parallel. This will typical be the case for the session chair during the starting period of the session (compare Figure 9), or at the end of the session in order to process multiple simultaneously received leave indications. At the same time it is possible that the chair will want to invite a number of users into the session, this is once again best done in parallel and not sequentially (i.e., one does not want to wait for an invite to finish before initiating the next invite operation).

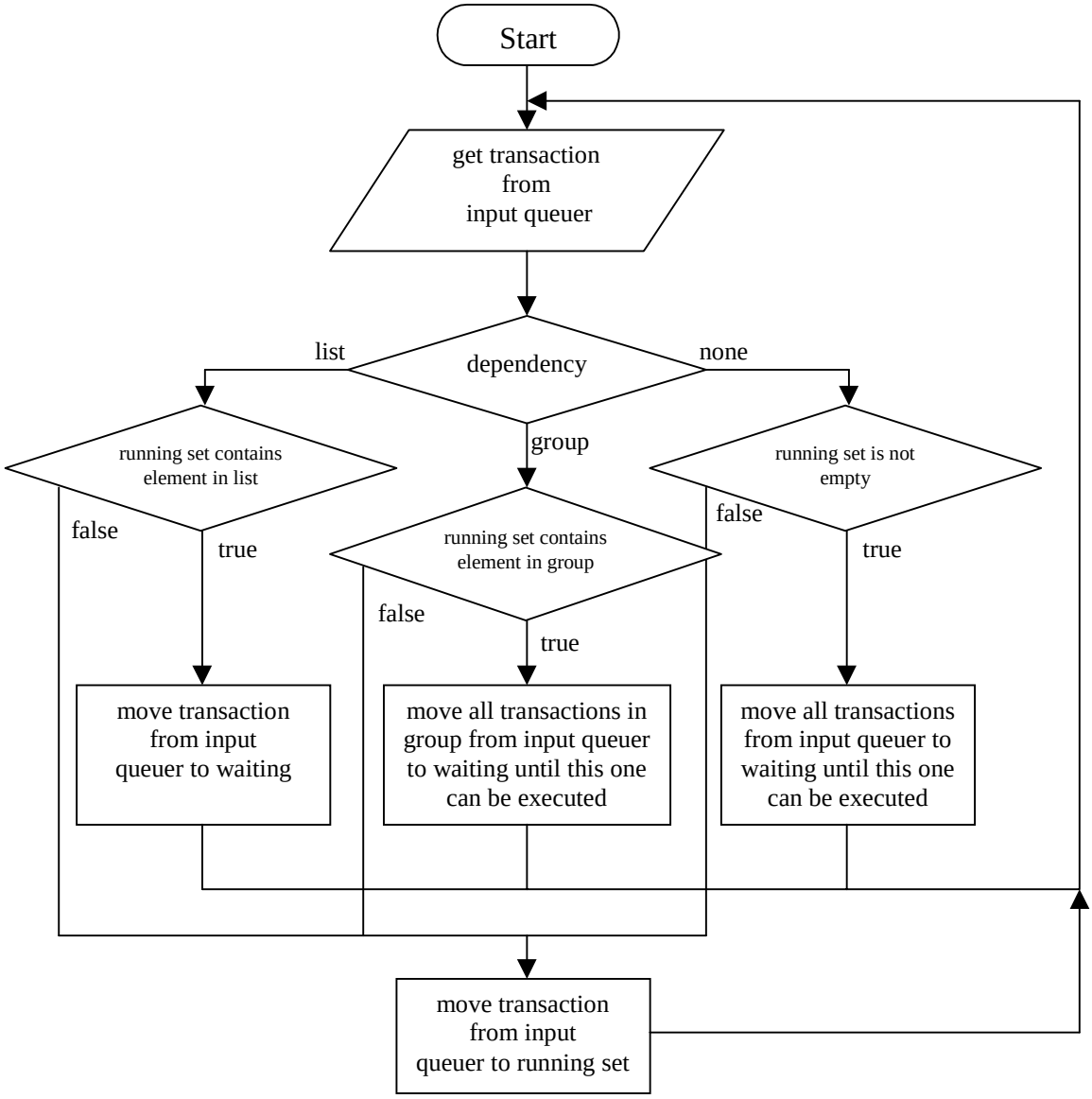


Figure 25: Core logic of transaction system (dependency checker)

In order for the transaction system to resolve the dependencies and to decide if a transaction can be moved to the running set the process shown in Figure 25 will be execute. For each transaction in the input queuer the independency type is determined. Based on this type a check for dependency conflicts is performed. If no conflicts are detected the transaction is moved to the running set. For the independency type **none** this will only apply if the running set is empty. If this is not the case all following transactions will have to be moved from the input queuer to the waiting set until execution of the transaction under consideration becomes possible. This is necessary as the type **none** is the most restrictive one and otherwise less restrictive types could constantly occupy the running set without it ever becoming empty. This is clearly not acceptable. For the independency type **groupnone** a similar situation applies, only that here the scope is limited to the specific group of this transaction and thus the check is only done for transactions of the same group contained in the running set. If any transaction of the same group is detected all following transactions belonging to this specific group will have to be transferred from the input queue to the waiting set. The remaining case is straight forward. For the independency type **list** the running set is searched for elements of the same group as the transaction under consideration and with a type/name contained in its dependency list. If such an element exists the transaction has to be moved to the waiting set.

This schema possesses an implied higher priority for more restrictive transactions (by independency type). This is necessary to prevent less restrictive transactions to constantly defer the more restrictive transactions. For the independency type **list** no further implied priorities exist (these could, for example, be defined by the number of entries in the list). It is therefore up to the system/transaction designer/developer to correctly adjust dependencies to prevent the previously described problem of repetitive delaying execution of transactions. This as well shows how important the correct configuration is. Only a correct configuration will enable a smooth and especially efficient performance of the whole system! Currently no automatic system is in place to perform these checks and thus they have to be done manually. For the in Table 17 shown configuration the following arguments verifying its correctness apply. The risk of conflicts is given by the structure of the databases, as shown in Figure 20, due to backward references. This is basically the case by the session entries pointing into the user and application database. Here a change in the user database might cause changes in the session database. If the session database is not updated references might become invalid. This results in the need for remotely/periodic initiated *SystemUpdate* and *ExpireUsers* to belong to independency group **none**.

The simultaneous access on a single database is only somewhat critical. Here it is sufficient to ensure (via language constructs, in the case of Java *synchronized*) that simultaneous write access are not permitted. Once an entry has been retrieved operations on it are independent of the other entries in the database. Thus, for transactions which modify fields on which further subentries depend (e.g., user references in session entry) it must be ensured that here no conflicts are generated. This can be achieved by setting these transactions to exclusive execution within their group (as indirectly defined by the database entry under consideration). As a result these transactions will use an independency of **groupnone**, as is, for example, the case for locally initiated *SystemUpdate* and *SystemAlive* in the system group, as well as *SessionCreate* and *SessionTermiante* in the session group (both for local and remote initiation). These are by their sequential dependencies forced to be exclusively executed within the group, as shown by Figure 23. The same applies for locally initiated *SessionJoin*, *SessionLeave* and *SessionUpdate*, as well as for remotely initiated *SessionInvite*, *SessionExpel* and *SessionExpire*.

This basically leaves the main row of interest which is focused on the parallel execution of received join requests and leave indications, as well as on locally generated user invites and user expels. These operations will all affect the user references in the session entry, as can be seen in Figure 20. Despite this a parallel execution is (partially) possible as a number of worksteps of these transactions can be executed in parallel. Only some worksteps have to be locked (synchronized) against simultaneous access. Taking the join as an example to illustrate this process:

1. A join request is received via the network and a *JoinSession* transaction is generated. This process is sequential due to the nature of the network (and the access to it by Java).
2. The join transaction will build a user entry, this includes some processing for hardware, thumbnail and access (implicit/explicit/filtered) information. This can be done in parallel as not access to any other resources is needed.
3. The session controller is asked to confirm the join for the user. This will basically be done sequential, as the session controller requester is synchronized, thus only allows to be executed once at a time. As this request does not include any processing it is a rather simple operation.
4. Now the response is build. Again this can be done in parallel.

5. Next the response is send. This will again be a sequential process, due to the same reasons as explained in 1.
6. The user is entered into the database, again this is sequential, as it is a write access of the database.

As a result one can see that even for these types of transactions some implicit parts require group wide sequential execution, but these are only limited to certain small worksteps. These only require this to access unique management structures, but the main point is these processes will not have an effect on other entries of the same type. Therefore certain operations on subentries of a session, as is the case for users in the session, can execute in parallel. Each of them is basically independent of the others and no situation can arise were one of them might use preliminary entries of another one to act upon. Thus no risk to cause a conflict if these preliminary entries are not confirmed, modified or even deleted by the other transaction exists. As a result one can define two basic categories for transactions according to the above reasoning.

- Those which are intra database dependency free (iddf) and
- those with multiple database access dependencies (mdad).

Hereby the latter may well be limited to a single database, but may as well concern multiple databases (database restrictions). As one can see the setup of independencies is a very cumbersome process requiring an in depth look at the system, especially at the relations between the involved databases and how these are accessed by the involved transaction. An automatization of this process could be very helpful, but is a difficult task, especially if not only the question of correctness of a configuration is to be determined, but as well the efficiency of a setup (in order to reach an optimal setup).

For the setup of the transaction system it is thus necessary to identify the basic category of each transaction in MACS which results in the classification as shown in Table 18. Thus, taking the sequential dependencies (see Figure 23), combining these with transaction independencies as defined via the basic categories of transaction (see Table 18) and the relevant database structures (see Figure 20) in order to determine the database restrictions one gets the configuration for the transaction system as shown in Table 17.

Taking as an example the *CreateSession* transaction. The sequential dependencies tell us, that it has to be executed after the initialization of MACS, but before any other transaction regarding this session. Therefore these can not be executed in parallel (which is in this case a rather obvious conditions). Now checking the independencies via Table 18 we get that *CreateSession* has multiple database access dependencies (mdad) regarding its session entry. This does not limit its parallel execution further, as only transactions which have already been excluded via the sequential dependencies have the same category and a fitting database restriction. Next considering the database structures (see Figure 20) one sees that the session entry is part of the session database and thus all transactions with a multiple database access dependency and a session database restriction are not able to execute concurrently with the *CreateSession* transaction. An example for this is the remotely initiated system update (*RecvSystemIndication*), which potentially accesses all databases and therefore might cause data inconsistencies in the session database.

transaction name	basic transaction categories (incl. database restriction)
SetupMacs	mdad multiple database access dependencies - all
ShutdownMacs	mdad multiple database access dependencies - all
EnterMacs	mdad multiple database access dependencies - all
LeaveMacs	mdad multiple database access dependencies - all
SystemAlive	iddf intra database dependency free
SystemUpdate	iddf intra database dependency free
RecvSystemIndication	mdad multiple database access dependencies - all
ExpireUsers	mdad multiple database access dependencies - all
CreateSession	mdad multiple database access dependencies - its session
RecvSessionCreate	mdad multiple database access dependencies - its session
TerminateSession	mdad multiple database access dependencies - its session
RecvSessionTerminate	mdad multiple database access dependencies - its session
SessionAlive	mdad multiple database access dependencies - its session
RecvSessionIndication	mdad multiple database access dependencies - its session
ExpireSessions	mdad multiple database access dependencies - sessions, user
JoinSession	mdad multiple database access dependencies - its session
RecvSessionJoin	iddf intra database dependency free
LeaveSession	iddf intra database dependency free
RecvSessionLeave	iddf intra database dependency free
SessionInvite	iddf intra database dependency free
RecvInviteSession	mdad multiple database access dependencies - its session
SessionExpel	iddf intra database dependency free
RecvExpelSession	mdad multiple database access dependencies - its session, user

Table 18: Basic transaction categories

The configuration of the transaction system, as defined above and shown in Table 17, is read from the transaction system configuration file. The structure and mechanisms used are similar to the system configuration. Thus it is possible to easily integrate new transactions and to adapt existing transactions to changes in the system. This is, as previously explained, a difficult task and might require a complete reevaluation and verification of interdependencies for all existing transactions in order to ensure the reliability, stability and efficiency of the whole system. Therefore, despite the accessibility of the configuration, it is not envisaged that anybody except the developers will modify dependency information.

3.4.5.2 Control protocols

A protocol was defined to allow the interaction among MACS instances. It is divided into three basic sections, these are:

- system messages
- session messages
- server messages

This division is in direct accordance with the structure of the communication services as defined in section 3.4.4.3. In the rest of this section the messages, as shown in Table 19, of each of these three protocol sections are introduced including some short explanation. A number of (interesting) messages are chosen and explained in more detail, while for the others the detailed descriptions are included in appendix A.3. Further information about the control flow within MACS with details for each message is contained in [92].

function	**type**	**details**
server delete	server	1 step indication
server invoke	server	3 step: request, response, indication
server leave	server	1 step indication
server request	server	3 step: request, response, indication
application control	session (application)	2 step: request, response
controller data	session (direct)	1 step: indication
floor data	session (floor)	1 step: indication
session create	session (direct)	2 step: request, response
session indication	session (direct)	1 step: indication
session terminate	session (direct)	1 step: indication
user expel	session (user)	1 step: indication
user invite	session (user)	3 step: request, response, indication
user join	session (user)	3 step: request, response, indication
user leave	session (user)	1 step: indication
system indication	system	1 step: indication

Table 19: Types of communication actions

Table 19 gives a overview of available communication actions, showing their function, the corresponding protocol section and some details of the structure. Here three types of structures are found:

- 1-step (indication)
- 2-step (request, response)
- 3-step (request, response, indication)

1-step actions are simple indication send to inform other participants about changes, for example, within the session. A 2-step action is formed by a request response pair. In such a case a direct response to the corresponding request is possible, as, for example, is the case for the creation of a session. Finally a 3-step action first sends a request, will then receive a response form the remote host and finally announce the changes resulting from this operation via a indication. These types of communication actions are independent of the protocol section.

System messages

The system messages, or in this case better the system message, is used to inform other MACS instances of the existence of the local MACS instance. Updates of the local information (e.g., change of user contact details) are distributed by this message. Its other function is to act as an alive indication for the other systems. If no (alive) message is received from a given system for a (via the configuration) determined amount of time, it is assumed that the remote system has technical problems or was separated from the net, thus leaving no direct connection. In such a case the information about this system is purged from the databases, including all sessions announced by the system. If the user of such a system participated in any of the ongoing sessions handled by the local system, the corresponding sessions will be notified of the detected loss of contact and, thus, drop the user (that is remove the user form the session). Figure 26 describes the message structure and its distribution.

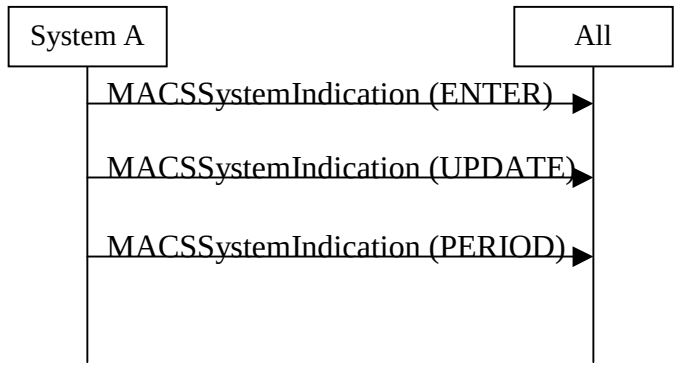
<table>
<tr><td colspan="3">Name: MACSSystemIndication</td></tr>
<tr><td colspan="3">Description: This message is used to send information about the local system to other systems:
• as an announcement when starting the local system (type ENTER)
• as an update for changes of the information about the local system (type UPDATE)
• as a periodic alive message (type PERIOD)</td></tr>
<tr><td colspan="2">Channel: global</td><td>Type: system</td></tr>
<tr><td colspan="3">Parameter</td></tr>
<tr><td>• sequence</td><td colspan="2">sequence number</td></tr>
<tr><td>• state</td><td colspan="2">system status (ENTER, UPDATE, PERIOD)</td></tr>
<tr><td>• user</td><td colspan="2">description of user</td></tr>
<tr><td colspan="3">Diagram
 
</td></tr>
</table>

Figure 26: MACSSystemIndication message

Session messages

Session messages are used to coordinate actions within a session and to distribute session relevant information among MACS instances. These can be further subdivided into four categories according to their respective function and content:

- direct session control
- user control
- application control
- floor control

The direct session control related messages are responsible for handling the basic existence of the session and include messages for its creation, termination, update and the direct communication between the involved session controllers. The terminate, update and controller messages are simple indications and their description is included in the appendix. The create message is described in more detail in Figure 27.

The create message is a two step process, consisting of a request and a possible response. The response is only send if the remote system detects a resource conflict. If during a preset timeout period no response is received the local system assumes that no resource conflicts exist and creates the session. Once the session is created a background process starts sending session indications. These are in their functionality equivalent to the system messages, but are bound to a specific session. The first session message has ENTER set as its type, afterwards for the periodic updates the type is PERIOD. If changes to the session details occur, as, for example, a change to the session description or the contact information for the session an UPDATE type indication is send.

The main resource conflicts which can occur when creating a session are session naming and address allocation conflicts. The naming conflicts are not really problematic, as sessions are identified by a unique identifier (formed via host address, user login and a timestamp). As a result two sessions with the same name, but different creators could actually coexist without problems, but this is not desirable, as in the session list multiple session entries with the same name are inconvenient to the user. Thus, if a create request for a session with a already (locally) existing session name is received a negative response is generated. An address conflict on the other hand will generally be a real problem, as in the case of a multicast group, for example, messages not belonging into the session will be received and might cause misinterpretations.

Name: MACSSessionCreateRequest, MACSSessionCreateResponse	
Description: These messages are used to create a new session. Other systems can reply to the create within a given timeout to prevent the creation of the described session (e.g., to prevent naming conflicts).	
Channel: global	**Type:** session
Parameter	
Name: MACSSessionCreateRequest	
• sender	identify sender
• sequence	sequence number
• session	session description
Name: MACSSessionCreateResponse	
• sequence	sequence number
• request	identify sequence number of request
• requester	identify sender of request
• reason	the reason why the session should not be created
Diagram	

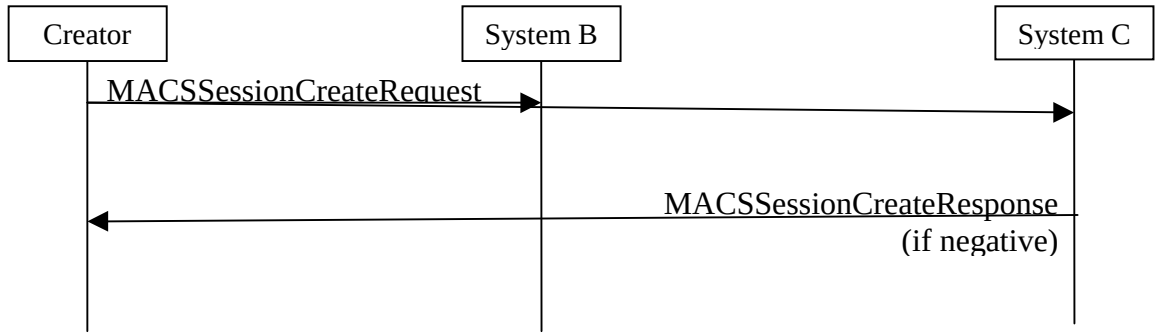


Figure 27: MACSSessionCreate messages

The user control related session messages are used to handle join requests or invite requests, as well as leaves and expels. Thus, they are in some respects similar to SIP [46] in that they serve to exchange session setup information. SIP will, for example, on a successful invite exchange setup information and the intention of the other person (that is, if he did accept the invite or not), but SIP is not involved in actually setting up the communication links or to pass status information on to remote hosts. MACS on the other hand assumes no further exchange for setting up the communication links to be necessary. Further, MACS exceeds the ability for session setup, invite and leave of SIP by offering additionally periodic status information, expel and floor control capabilities.

In MACS the join and leave are complementary in their design to invite and expel just with the active (i.e., the initiator) and the passive (the user in question) parts exchanged. For invite and join two types exist:

- direct
- delayed

The *direct* type assumes that the session in question is open or has an implicit (see section 2.2) access control, that is that the decision whether to allow the requester to join or to accept an invite can be determined immediately without having to interact with the session chair or the user via a dialog. In such a case join or invite is a request and response message pair, followed by a session indication once the process is concluded. This is done to inform the other systems in the session about the changed participant list.

For the *delayed* case an interaction with the session chair or user becomes necessary. Here a response indicating a delay is send. This leads to marking the sequence number of the originally send request as a delayed message. The other part will now initiate the interaction with the session chair or user. If the result of this interaction is negative a response is prepared to indicate the negative outcome to the requester. As a result he will then delete the marked request sequence number from his list of delayed messages. On a positive outcome an invite or join with the previously marked sequence number is prepared and send by the system which did interact with the user or session chair. This request will contain the sequence number of the earlier request leading to the delay and thus will be directly accepted by the other system. Figure 28 shows this in detail for the join case. Thus, for the delayed case a pair of join/invite requests will result, with the following two possible sequences between the hosts A and B:

- join
  1. A: join request to B
  2. B: join delayed to A
  3. B: invite request to A (marked with sequence number of previous join request)
  4. A: invite response to B (automatically accepted, as recognized by sequence number)
- invite
  1. A: invite request to B
  2. B: invite delayed to A
  3. B: join request to A (marked with sequence number of previous invite request)
  4. A: join response to B (automatically accepted, as recognized by sequence number)

A complete example of exchanged messages for a session setup and a delayed join can be found in appendix A.3.3.

Name: MACSSessionJoinRequest, MACSSessionJoinResponse	
Description: This message is used to join an existing session. The remote control will check for (control) resources and if these are available pass the request on to the remote session controller.	
Channel: session	**Type:** session
Parameter	
Name: MACSSessionJoinRequest	
• sender	describes the sender
• sequence	sequence number
• request	sequence number of earlier invite request, if applicable (else 0)
• user_data	additional user data (optional, e.g. for Visco)
Name: MACSSessionJoinResponse	
• sequence	sequence number
• request	identify sequence number of request
• requester	identify sender of request
• reason	reason code (OK, or an error code)
• user_data	additional user data (optional, e.g. for Visco)

Diagram

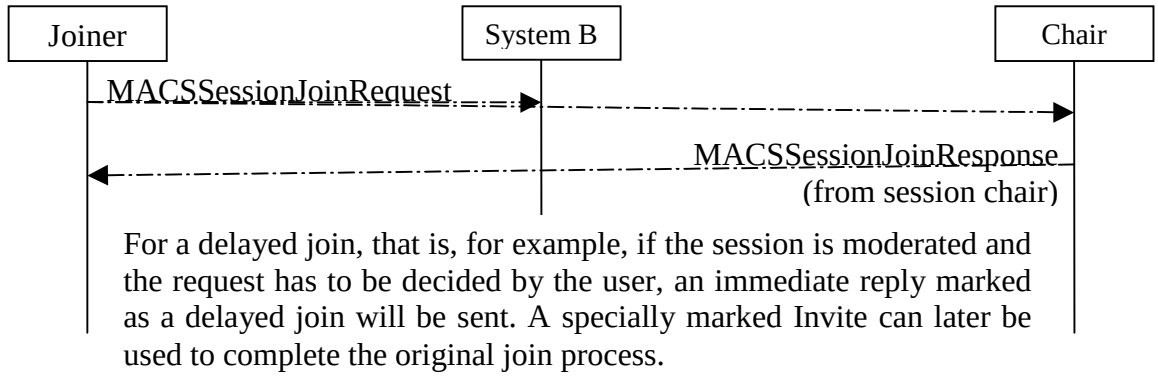


Figure 28: MACSSessionJoin messages

The session messages related to the application control consist of a single request/response pair. It is used to indicate the start of an application within a session and for generating an address for this application. Indirectly the session indication message is involved as well, as it will indicate to other systems if an application is stopped, thus, making an extra indication for this purpose unnecessary. The details of the application control message are shown in Figure 29.

Name: MACSSessionApplicationRequest, MACSSessionApplicationResponse	
Description: These messages are used to coordinate the start of an application in a session, especially the assignment of an address for the application to use.	
Channel: session	**Type:** session
Parameter	
Name: MACSSessionApplicationRequest	
• sender	describes the sender
• application	describes the application
• sequence	sequence number
Name: MACSSessionApplicationResponse	
• sequence	sequence number
• request	identify sequence number of request
• address	address assigned

Diagram

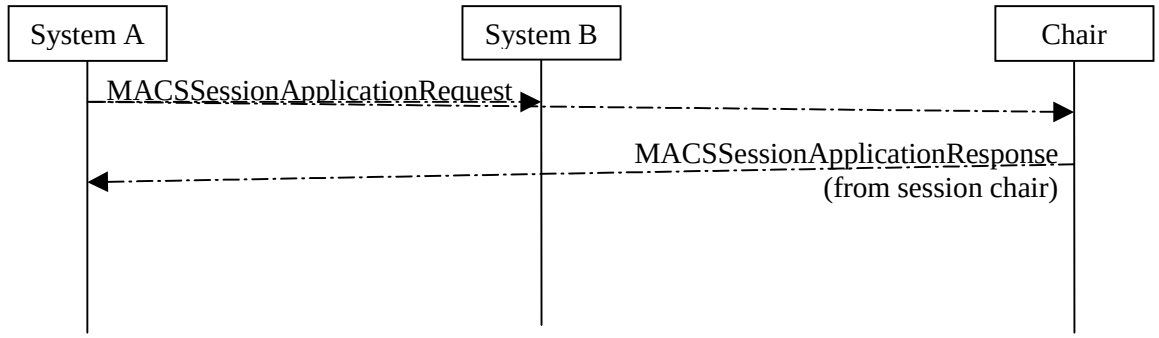


Figure 29: MACSSessionApplication messages

The session messages concerned with floor control, or more exactly the floor indication message, is used by the floor control (see [19]) to send floor data. The message is a simple indication.

Server messages

Server messages are used by the group of server setup [42] and can be grouped into three basic functions:

- invocation
- deletion
- request passing

If the system determines that a new server is desirable (or a manual configuration is available) a MACS instance is selected to become an additional server for the session. The selection process is a potentially very complicated task and described in [42]. The selected system is then informed via an invocation message to activate its server support. Once the server

support is activated a number of MACS instances might be transferred to the control of the newly invocated server.

The deletion message does the opposite of the invocation message, that is it removes a server from the server group. If a server receives such a message it will transfer its clients to another server and become a client to one of the other servers itself. Finally it will deactivate its server support. The reasons for removing a server from a group can differ quite substantially. It is, for example, possible that all its clients have left, making it unnecessary to continue operating as a server. Further it might be the case, that the network connection quality has changed to such an extend, that it is no longer useful for a system to act as a server. In this case a different MACS instance might be more suitable to take over its role. These are all cases caused by external signaling to the server, but two other cases exist.

The server can actually decide itself to leave the session, thus making the transfer of its clients to another server necessary. Such a process could be initiated through a leave indication. Further, the server can experience a direct (e.g., system crash) or indirect (separation from the network) failure. In this case the other servers must detect this condition and resolve it. This can potentially be rather complicated, as the clients serviced by the server have to be moved to another server. Due to the abrupt failure of their original server this can not be done in a preorganized way, leading to a fault recovery situation. The normal case would assume the availability of the old server until the transfer is concluded. Especially considering faults due to network separation the failure condition might equally well concern the clients handled by the separated server and thus make their reintegration into the session and their continued servicing via a different server impossible (as they are too separated from the other servers due to the network failure).

The server request functionality is in comparison straight forward. A number of situations can not be directly resolved by a given server but might need, for example, the involvement of the session chair. The session chair will, in general, be represented by one of the servers belonging to the server group. Thus, if a server has to handle such a request no other possibility exists but to pass the request on to the chair and wait for the result. This is done via a server request/response message pair.

3.4.6 Design evaluation of the MACS control level

Two easily identifiable limitations result from the here introduced design:

- client server architecture (except for group of server extension)
- transaction system is a potential bottleneck

Classic client server technology is well understood and its extensions, as, for example, the approach of using a group of servers, allow an increase in the achievable performance and the overall availability (i.e., fault tolerance) of such systems. The MACS framework uses these techniques to increase its scalability in a modular way.

The transaction system is a potential bottleneck, but will deliver a much better performance than a simple locking approach for the databases, as discussed in section 3.4.3. The sequential case for the database access is an even more severe bottleneck. The transaction system enables an efficient usage of the available system resources and helps to increase the responsiveness of the system, while at the same time increasing its maintainability by providing dynamic configuration and allowing the extension of the system in an easy to handle way.

The overall design thus achieves a high level of modularity and hence a degree of flexibility hardly found in any other system, especially not in conjuncture with the advanced services offered, such as, for example, floor control.

3.5 GUI level of MACS

The Graphical User Interface (GUI) provides the interaction point with the user. Here it is very important to design an interface which is easy to use, but still informative and powerful. This document covers, as before state, only the aspects of the control visualization, but not the visualization of the session controller, which is covered in [19].

First the requirements for the visualization are analyzed and then design goals are derived. Next a short overview of relevant scalability aspects is given. Finally the structure of the GUI level is explained and some advantages and limitations in the design are identified.

3.5.1 Requirements and design goals for the GUI level

As MACS possesses a modular structure the GUI level has to be modular as well. To achieve this the MACS window has been divided into a number of sections providing screen space for different components of MACS. These are integrated into a space saving, but easy to understand layout [32] allowing a high degree of flexibility and scalability. The requirements and resulting design goals are therefore:

- modularity
- flexibility
- scalability
- easy to use

3.5.2 Scalability aspects of the GUI level

As mentioned before screen space is an important limiting aspect. Each application therefore should use the minimal amount required to provide an easy to use interface. Normally only a limited number of tools are used in any given session. The control elements of these are, as previously described, assembled into groups by the MACS control. If these require more space than available scrollbars for the specific areas are enabled. This is unfortunately not very user friendly, but if the user really requires the requested amount of screen space he can scale the control window and, thus, reduce or even remove the scroll areas.

3.5.3 Structure of the GUI level

The GUI level of MACS is divided into a number of groups represented on different tabs. Tabs were chosen as certain functionality is unlikely to be used simultaneously and therefore has not to be simultaneously visible. The default tab provides access to session data. A further tab does the same for the user data. Two configuration tabs exist, which can optionally be displayed (and hidden again). These allow access to the configuration (controlling the content of the configuration file and setting of user preferences) and to the modules. For each joined session a further tab is created. Thus, the tabs and their content are as follows:

- session list tab
  - session list
  - session detail
- users list tab
  - user list
  - user detail
- modules tab (optional)
  - module list
  - module detail
- preferences tab (optional)
  - preference key list
  - preference key values
- session tab (one for each joined session)
  - session controller
  - application launcher
  - application controls

The session list tab contains a list of all currently known sessions. These are presented in an alphabetical order within their groups, as shown in Figure 30. Here two groups of sessions exist, MACS sessions and sessions received via the MBone interworking module. To allow the user to easily distinguish between them they are marked by different background colors (MACS grey, MBone white). As the aim of the session list is to allow the inspection of existing sessions the right part of the tab contains the details about the selected session, including name, description and contact information of the session creator.

Figure 30: Example of a MACS session list tab

The user list tab is similar to the session list tab. The design aims at providing easy access to the list of known users and detailed information about these, as seen in Figure 31. Here an important issue is to provide not just a name and the system specific contact information, but as well their available multimedia capabilities (hard- and software). This is important as the user can thus, before establishing contact, determine which types of communication (textual, audio or video) are available. The MACS system further allows to transmit a thumbnail, a feature which can help especially in a situation were the users have met before, but do not know each other well. This is further useful later on in the session in order identify the other person (as explained for the actual session tab). Again, in the user list the MACS entries are sorted in an alphabetical order, with exception of ones own entry which is always displayed at the top position of the list. A coloring schema is used again to identify the different groups of entries. Ones own entry is marked with a yellow background, the entries of the MACS users are marked with a white background and the others, added via the ITU interworking module, are marked with a grey background. MBone users could as well be added to the list using another background color to distinguish them from MACS and ITU users. As typical MBone tools do not distribute directly user information this approach will only be feasible for already joined MBone sessions, as here the users can be extracted relatively easily and added to the user list. For MBone users in sessions currently not joined no information will be available.

Figure 31: Example of a MACS user list tab

As MACS allows the integration of modules into the control, as described in section 3.4.4.2, it is important to offer the user an easy to use way of accessing active modules and to determine their status. On the other hand is this an advanced feature and should therefore not be presented by default to all users. This is solved by providing an optional tab which can be activated via the toolbar. This tab, as seen in Figure 32, contains a list of the locally known modules on the left hand side and details about a selected module on the right hand side. It is thus in its structure similar to the two previously introduced tabs. One main difference is that this tab can be closed again and hence removed from the tab bar. It is important to note that all locally known modules and not just the activated or running modules are shown.

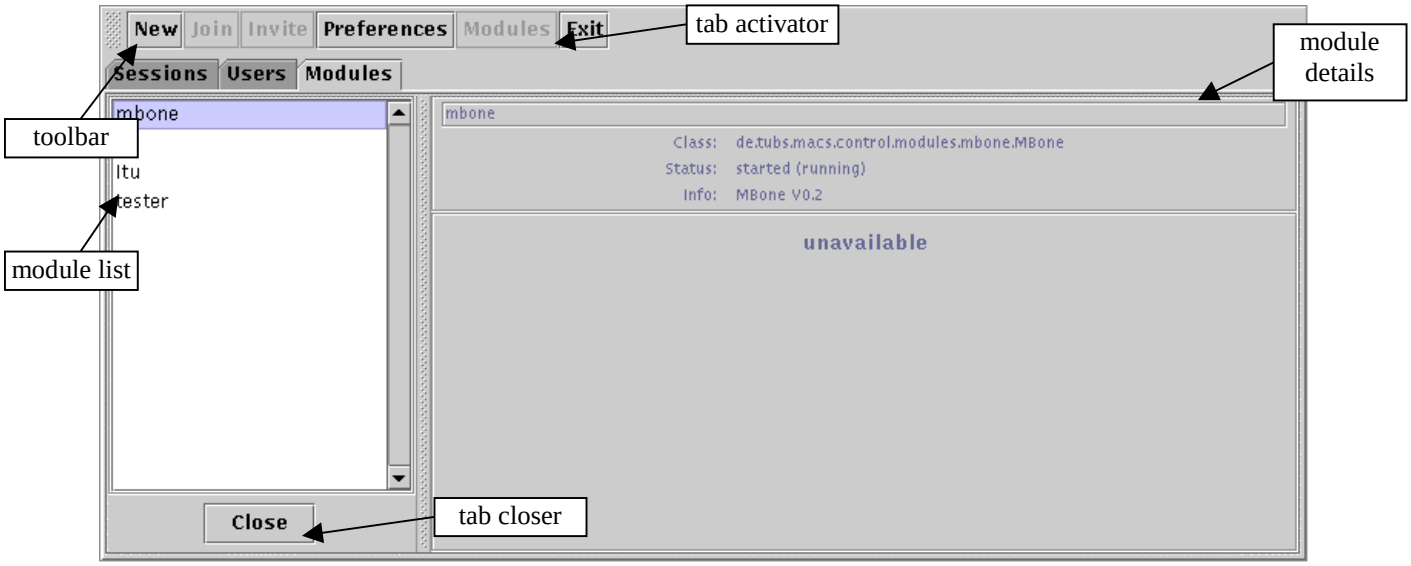


Figure 32: Example of a MACS modules tab

The preference tab allows access to the configuration files of MACS and, thus, is very important. For a flexible and modular system easy configuration is a key factor. The tab is designed similar to the previous tabs, thus providing a consistent user interface. The left hand side gives a list with a selection of configuration sections, while the right hand side provides details for the currently selected section, as shown in Figure 33. A key difference is that here the right hand side is not just a display of the current configuration, but allows the user to customize his setup by overwriting the system specific pre-selection. These modifications of the defaults are stored in a user preference file in a for the userfully transparent way. The example below shows how the identity configuration section of the user looks like. If the user wants to reset the section to the default values, for example, due to an incorrect modification, he can use the **default** button. The **save** button allows him to store his preferences, while the **close** button allows him to close the tab. This design allows an efficient and user friendly access to the system and user specific configuration, both for novice and expert users. Expert users can as well chose to use the "All" section to access all available configuration options.

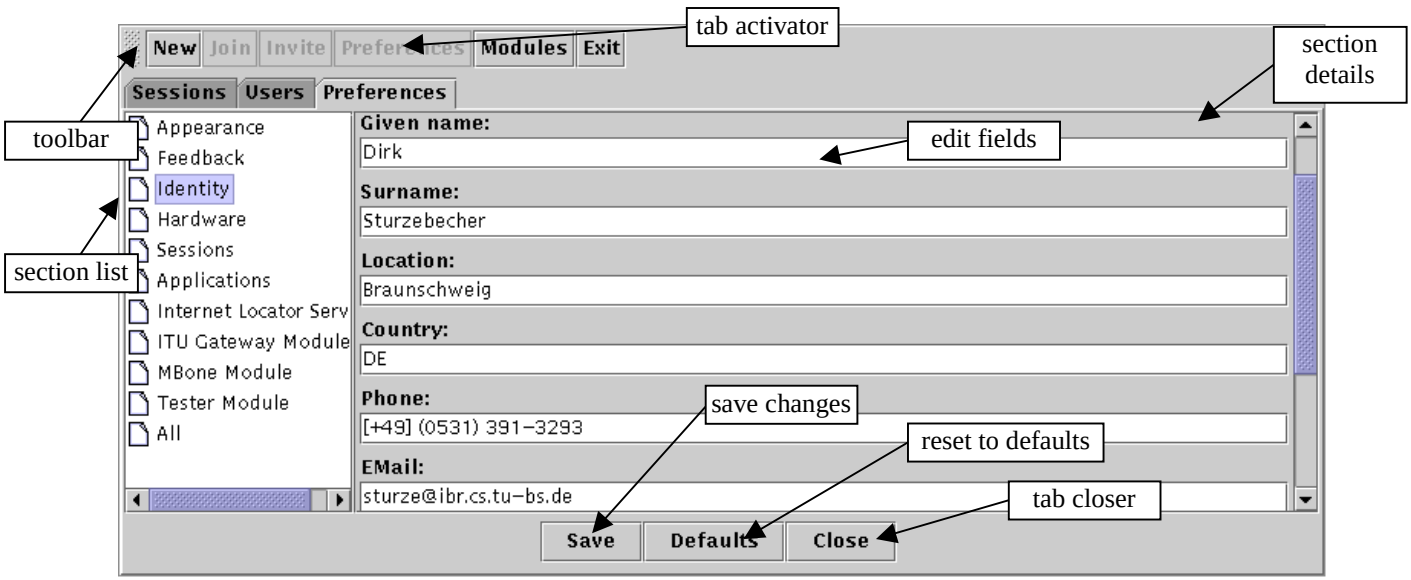


Figure 33: Example of a MACS preferences tab

The preference tab is, as the module tab, optional and as such can be activated via the toolbar and hidden via the **close** button. For a modular and flexible system as MACS, especially considering the easily possible integration of new modules and, thus, configuration options, the preference system and its visualization have to be very flexible. This is achieved by setting up the configuration editor via yet another configuration file (see appendix A.4 for details). This text file can easily be edited by the module developer to include new configuration options.

Coming back to the visualization. The most important aspect of visualization for the system is the visualization of the actual session. Each session is displayed in its own tab. The tab is divided into two basic areas, as shown in Figure 34. The controller part is on the left hand side and provides the most important part of the feedback about the ongoing session. The controller has to chose a representation fitting the current session type and the given usage scenario. The actual visualization of the controller is outside the scope of this document and described in detail in [19]. The right hand contains the application area. Here the control elements of all applications are displayed (e.g., the audio tool elements). Further, for certain application modules requiring only a very limited amount of space a area to directly display all of its components can be arranged within the application area, as shown for the chat module. The lower part of the session tab contains the launcher bar. It allows to start the application modules available for this session. A further important feature is the feedback about application modules used by others [16]. This is needed in order to easily establish communication between the participants (e.g., it is not useful to start a chat tool if all others are only using the audio tool).

Figure 34: Example of a MACS session tab

3.5.4 GUI level design evaluation

The here introduced design guarantees a high degree of flexibility, by allowing the selection of, for example, a session controller specifically suited for the planed session. Further are the screen space requirements kept quite small, which is important, as screen space is a rather limited resource for a CSCW system. This is achieved by splitting the information in such a way, as to group it by (for a CSCW system) typical tasks. Still some small limitations do exist. This design will cater well for sessions with an average amount of applications and a user normally only actively participating in a single session. Specific cases might exist were a larger number of applications is in use resulting in very large screen space requirements for

the control window. On the other hand it can be assumed that these applications actually do display data and interact with the user in some way, thus probably requiring even more screen space to display their results. The support of active participation in more than one session at the same time is somewhat limited, as in MACS only a single session tab can be visible at any given time. It is here necessary to first analyze if such a behavior is actually useful and if it should be supported. Further, the support by applications with unique local resources, as, for example, audio, for such simultaneous participation is an open question which would need to be investigated before handling the corresponding control visualization issues.

3.6 Extending scalability via hierarchies

This section discusses a further scalability aspect not directly corresponding to a single level within the MACS framework and, thus, treated separately. The concept of hierarchies has been mentioned in the control level section, but really needs to be integrated into the complete system to provide maximum benefits. This section will treat the topic in more detail, by first introducing the concept, followed by a discussion about the current integration into MACS. Finally this section concludes in an analysis of requirements for improved support of hierarchies and therefore improved scalability of MACS itself.

3.6.1 Concept of hierarchies

The main factor limiting scalability is the number of active participants in the session. These are the participants fully known and controlled (i.e. tightly coupled) by the system. Inactive participants on the other hand will have a much smaller influence. This is true especially for large broadcasted session, where the number of (anonymous/inactive) listeners is basically not important. Each active participant has to be completely known to the system and administrated by it. Therefore the number of active participants will determine the size of the database (listeners need not be entered), as well as determine the complexity of the required control mechanisms. Looking at real-world conferences it seams that there are no situations which require a tight coupling of a large number of participants. A human being can not deal with thousands of persons at the same time. In order to form a group, such as the (inactive) listeners in a conference, it is sufficient to use loose coupling. A listener who becomes active can be included for the relevant period into the tightly coupled subset (assuming that the fluctuation of active participants is as in a real world conference rather small). In this way different levels of hierarchy (up to a complex structure) can be established. Only the moderator and the number of active participants will significantly influence scalability, while for the inactive participants a type of (anonymous) broadcasting (as commonly found in the MBone) is provided. Such a schema will allow the realization of very large sessions (podium discussions) with many hundreds/thousands of listeners.

3.6.2 Integration in MACS

MACS by its modular design allows the integration of this concept. This requires some minor changes in the management of the databases, as it is no longer guaranteed that each entry is complete. As a result it can happen that little to no information about passive participants is available. The session controller has to be modified or a new version created which is especially suited for the podium discussion scenario. The podium discussion scenario is regarding hierarchies the scenario of interest. The main problem is the support by the network and by the available network protocols. Only if these do support this hierarchy concept real advantages can be gained. If this is not the case only small differences will results as the data would then actually be distributed to all participants. Here it would then be ignored if the hierarchy level of the participant does not correspond to the level of the message received.

This will not provide the wanted reduction in overheads as would be the case for a properly supported hierarchical setup.

3.6.3 Requirements for integration of hierarchy levels

As stated above the most important requirement is the support by the network for this concept of hierarchies. Here a true multicast network infrastructure is fundamental to achieve a high degree of scalability. A protocol reflecting the suggested division in hierarchies will further increase scalability. Such a protocol can use different strategies for the different levels in the hierarchy and thus provide an optimal support for a hierarchical setup. At the highest level the messages have to be received as these are necessary to keep the system functional (i.e., for the servers used to control and serve the session). For a possibly existing secondary level the delivery of the messages is important and, thus, some effort should be spend to recover from transmission errors. On the other hand these message are not necessary for the functioning of the system itself. The results of a failure to deliver such messages may be a temporary incorrect session representation, such as, for example, displaying a no longer existing user in the user list. In such a case the inconsistency will be detected when a higher level operation is started to, for example, access the given entity (e.g., an invite). The system will then detect that this user is no longer present and correct the database. The lowest level, normally formed by the inactive participants, the listeners, will only get a simple best effort service without any error recovery. This might lead to some misrepresentation of session state or the inactive listeners missing minor parts of the presentation if the load on the network is sufficiently high. Still this setup will allow a very large numbers of participants. Again, if any listener becomes an active participant, he will be promoted to a higher level and, thus, the misrepresentations which could cause communication problems should disappear (or at least be drastically reduced).

An efficient network support is fundamental for a CSCW system. To achieve the high level of scalability while maintaining control over the session certain requirements have to be placed on the network. Only if these are fulfilled it is possible to fully support all scenarios as defined in section 2.2. The two main points not sufficiently supported today are reliability and QoS.

Reliability aspects

To efficiently support sessions with different hierarchies (e.g., moderator, podium, listeners) the network level must be able to support different levels of reliability (see as well 2.4.2.1) based on recipient groups for a single message. That means there will be a group of people who have to get the message or if a failure occurs this has to be directly detected. For the second group the delivery of the message is still important, but not fundamental. The system should try to ensure the delivery, but occasional failure is acceptable (semi-reliable). While the last group still is interested to receive the message it is not sufficiently important as to spend any effort accept the unreliable transmission (unreliable).

QoS aspects

Further it is necessary that the network supports different priorities. The control data has the highest priority and should never get lost due to bottlenecks or excessive network usage caused, for example, by audio or video data. Other applications might be able to work well even with bigger delays as acceptable for (non interactive) audio and video and could therefore use an even lower priority. QoS is an active area of research on many levels. Some organizations are, for example, trying to provide QoS tables for media used in CSCW sessions [1]. Further, research regarding the perception of different media [76] has identified

various factors influencing perception and determined threshold values in the perception of such media, thus, allowing to determine more easily the required parameters for achieving a good application support.

4 Application modules

A number of applications exists which are typically found in a CSCW system. These are, in general, an audio, video, chat and whiteboard tool. As these cover a very wide range of requirements, from high bandwidth and low delay for the video to low bandwidth and high reliability for the chat tool, each has different design issues. A further rather useful tool, but not as common, is a feedback tool. It allows to rate resources and events. Further it may even provide the remote support team with locally generated technical feedback and thus allow a detailed technical assistance. Such a tool can greatly increase the lecturers ability to assess the situation during a lecture based on the ratings provided, while at the same time allowing technical support of the session by an assistant.

The following sections will discuss in more detail some of the tools available for the MACS framework, showing how these can be efficiently integrated into the MACS framework despite their vast differences. First the media tools, that is audio and video are discussed, followed by the chat, whiteboard and feedback tool.

4.1 Media (Audio and Video)

Media support is a fundamental and difficult issue. Media support generally is formed by an audio and a video tool. Audio is required for cooperative work, while video can be optional, though in meetings and lectures it helps to identify oneself with the currently ongoing session. Both share the network requirements for a small delay and little jitter, though the video will require a significantly higher bandwidth.

First the functionality required by these tools will be analyzed. Then scalability issues are discussed. Finally the structure of each tool including the integration into the MACS framework are described. The section concludes with an evaluation of the design.

4.1.1 Audio and video functionality

The audio tool normally has a rather minimal GUI, providing only a volume control. Internally the audio tool is more complex, though most topics are more relevant to signal processing theory (e.g., automatic recording adjustments, noise suppression, silent detection and audio compression) than to the framework and application module design. These are therefore not covered in the scope of this document.

Despite this a basic description helps to better understand the system. The audio tool mixes incoming audio streams to form the overall "noise" normal to any real world classroom (at least during discussions or just before/after the start of the lesson). Advanced features might exist to modify the default mixing behavior. Mixing is as well relevant not only on a participants per session basis, but as well between sessions. The audio tool should support silence suppression and efficient audio coding to reduce network requirements/load. The speaker must easily be able to turn off his micro or mute his audio tool, while always having a clear feedback of the current state.

Similar arguments as listed for the audio tool do apply for the video tool. It should be easily possible to select if one wants to send video or receive other persons video streams. Controls as, for example, brightness and contrast should be accessible, but are of a lesser importance and therefore need not to be shown permanently. As the reception of a number of different video streams is a commonly wanted (though normally not supported) feature for most scenarios, an efficient display layout of these video streams is an important topic. The speaker

should be easily identified. One possible solution is the arrangement shown in Figure 6 with the speaker highlighted by displaying him in the middle, while others are displayed in an arrangement around him. Many other layouts are possible, but the analysis of these is not in the scope of this work.

Overall the following topics result for the audio and video application modules:

- audio
  - volume and mute control
  - microphone control
  - clear feedback of current settings
  - minimal network requirements/load
- video
  - easy control over send/received streams
  - efficient layout for multiple streams

Especially for the audio tool the floor control features are of relevance. Given a classroom scenario (as defined in section 2.2.2) it is desirable that the ability to transmit audio is only granted on request. Therefore, if the teacher calls a pupil to the front, the system should automatically assign certain floors to the pupil. This should at least be the audio floor allowing the pupil to speak, and might as well include the floors of other applications, such as, for example, the floor required to draw on the whiteboard. For this purpose MACS provides a floor group which includes as a standard the audio and whiteboard floors. This selection can easily be modified via the configuration editor to include other floors or to exclude a current one. A typical floor which could as well be included, if bandwidth allows, is the video floor.

4.1.2 Scalability aspects for audio and video

Audio places some load on the systems but for video the situation is much worse. Any system having to handle a number of audio and video streams at the same time requires a large amount of processing power to handle coding/decoding and rendering. Further load is caused by the network if no native multicast is supported, that is the data has to be send separately to each receiver. With native multicast support the sending of one owns streams to a single receiver is not at all different from sending them to multiple receivers, regardless of their number. This allows a much better scalability.

Both for audio and video it is not possible to present a large number of simultaneous streams to the person working with the system. No one is able to listen to many different things at the same time or to follow a large number of different video streams. This reduces the scalability problems somewhat, but requires an intelligent and efficient rendering of the received streams, that is mixing for audio and a corresponding window layout for video.

As the development of these audio and video features is not the core of this work the system relies on the capabilities provided by JMF [94] and assumes an underling multicast network.

4.1.3 Structure of media tools

The media tools are implemented as separate tools currently only linked via the control. The control offers different services, of which the two tools mainly use the floor control and the

display for their control elements. Further, a separation between sender and receiver exists. This allows to selectively start the required number of senders or receivers. Thus, one would start no sender if one does only want to listen or watch a session. This might as well be the case if one is unable to send this media (e.g., video) but can receive it. On the other hand one might start multiple sender instances. For video one can easily imagine a scenario with at least two cameras, one, for example, aimed at the local lecturer/speaker and another one aimed at the auditorium or at a product under discussion. This approach of separated receivers and senders allows a high degree of flexibility. The data is directly send via a RTP based network access point. Figure 35 shows the above introduced structure and its integration into MACS.

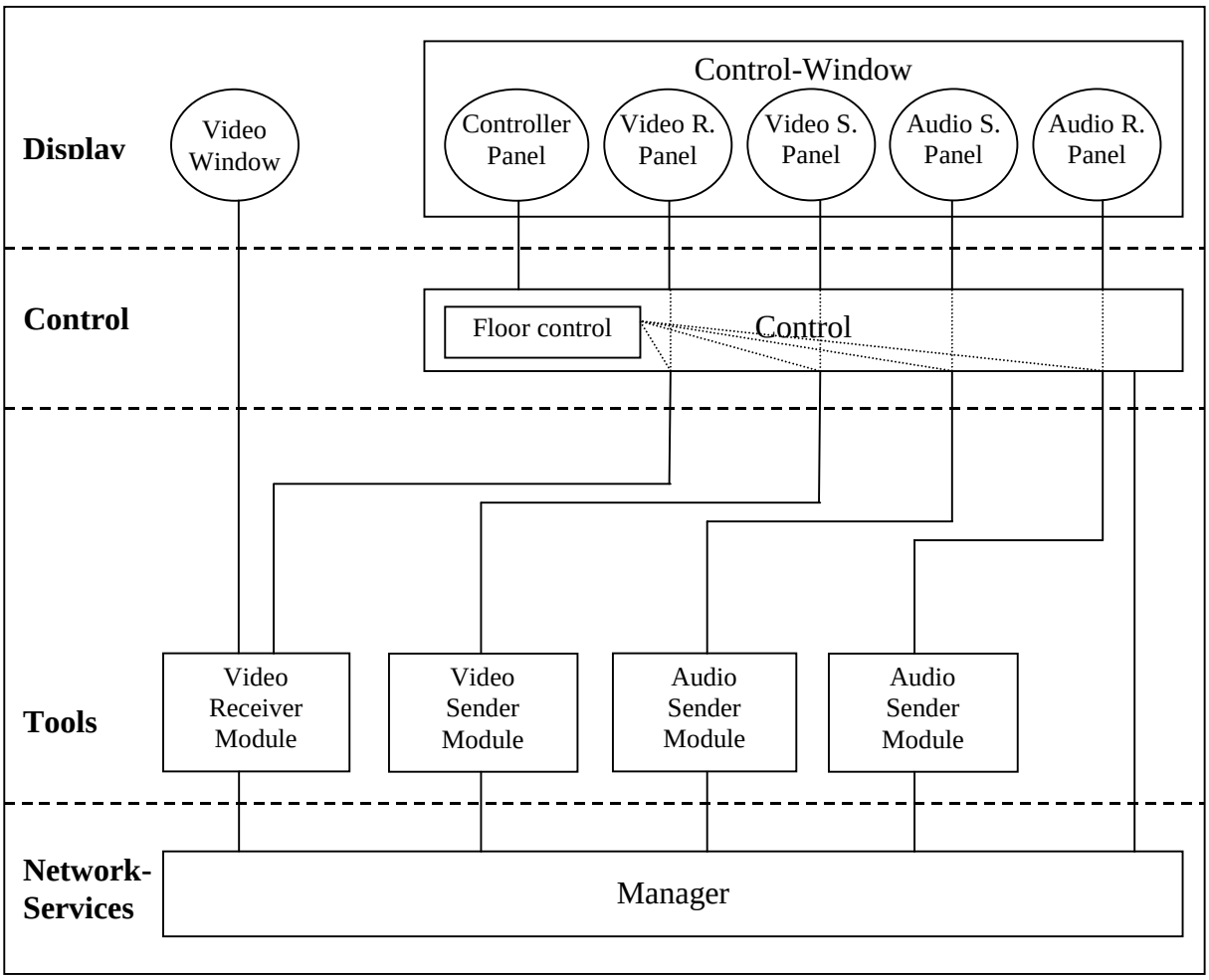


Figure 35: Integration of Media tools in MACS framework

The internal structure of the media tools is shown in Figure 36 for the example of a audio sender and receiver. Each of these contains a functional block for the GUI representation (in the control panel) and one for handling the floor control. As the floor control can be influenced through the GUI these two are interconnected.

Both the sender and receiver depend on JMF for providing audio capture and playback capabilities. JMF uses a typical setup from the music industry. To capture data a recorder object is instantiated and bound to a specific device (e.g., the microphone). This data is then written into a track. Hence a track is the recorded information from one specific source. A number of different tracks can exist at the same time. For music this would normally be a track for each singer and one for each instrument used. These are then later mixed to form one

composition. In JMF each track can be played back with a player object which takes a track as input. As all players share a common resource, the speakers connected to the computer system, a mixer will be used to combine the output of the players into one composition (i.e., the resulting sound/noise coming out of the speakers). The number of tracks which can be simultaneously played back by the players is limited by the number of mixer inputs in JMF. This is an internal value of JMF which is in Version 2.0 set to 32. This is sufficient for the use in a CSCW system (32 people speaking at the same time should surpass the ability of a normal person to understand).

Thus, a recorder for the audio sender and a player for the audio receiver exist. These are requested from JMF and then connected up to a track. The track in turn is connected to a transmitter (which can be a receiver as well) within JMF. Unfortunately it is only possible to influence the system during initialization and setup, that is connecting the recorder/player with the track and the track with the transmitter. It is afterwards not possible to directly access the data flowing through this pipeline structure, only a starting and stopping functionality is provided. For further detail see [94].

With this setup a simple, but complete audio tool can relatively easily be implemented. Further aspects, as, for example, audio coding are outside the scope of this work and hence not treated any further (except for using the functionality provided by JMF).

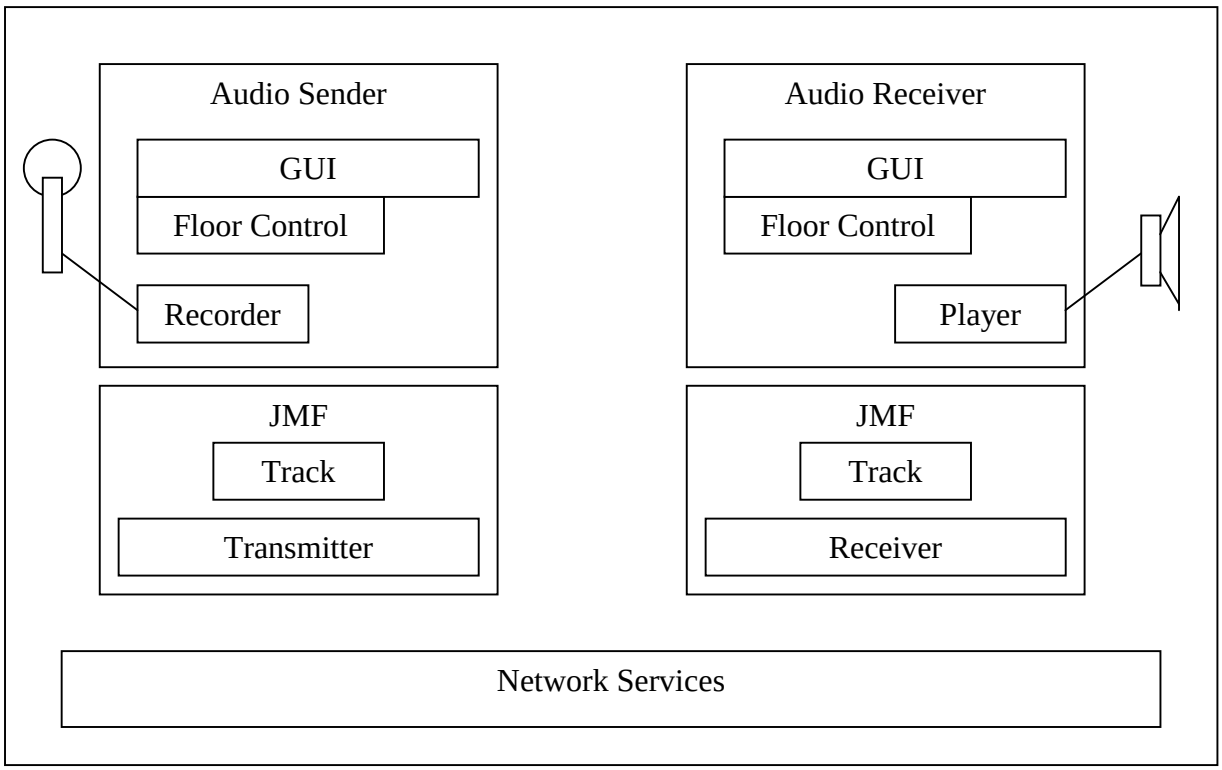


Figure 36: Media tool structure (audio example)

The configuration of the system is done via the normal MACS mechanism for system and user configuration, as previously explained

4.1.4 Evaluation of media tools design

The separation of receivers and transmitters allows a high degree of flexibility, making it possible to optimally adapt to the setup at hand. Further, a high degree of portability is gained by the reliance on JMF, as it allows a much more efficient and reliable development of a fully portable system. On the other hand the usage of JMF results in a number of limitations. These are mainly based on the early state of development of JMF and are, therefore, not further regarded at this point.

Some topics are at this stage not sufficiently covered by the media tools. The limitations due to the early stage of development of the audio tool itself are as follows:

- no synchronization
- only basic system/application integration (multiple access to single resource)
- missing media combination

To provide a consistent view on a session synchronization mechanisms are necessary. These can easily be integrated in MACS via messages send through the local control. In the case of audio/video synchronization mechanisms provided by JMF can directly be used, once these are properly supported by the available JMF implementation. But not only audio/video synchronization should be considered, but synchronization between all tools and the session controller itself should be enforced.

The audio and video tools are part of a single session, but do always refer to a single system resource, that is the camera, microphone or speakers. As soon as more than one session is active, a service is necessary to manage the requests by the different media tools for these unique resources. This can currently be done via the module tab, directly accessing the media module. This approach is not really easy to use. A better approach is needed, not that much concerning the structure, but especially providing an easier to use interface for the control and management of such resources. Thus, the media module needs to be extended to provide an easy to use interface within the session to resolve conflicts, better it should posses enough intelligence to resolve common situations autonomously.

4.2 The chat application module

A chat tool is used to exchange short text messages. This is a rather simple task and therefore chat tools are often used to test basic functionality. The following sections discuss the non testing related aspects of the chat tool of the MACS framework. First a short description of the functionality is provided, followed by some scalability issues. Then the structure is discussed and finally a short evaluation of the design is presented.

4.2.1 Functionality of the chat application module

The exchange of short messages via the chat tool is a simple form of communication and does only need minimal resources. In a session offering multimedia support only a very limited set of actions will typically be executed via a chat tool. It can, for example, be used to share web addresses with others during a presentation. As well it is used by a technical assistant helping a participant to overcome tool difficulties (e.g., of the audio tool). Floor passing is a possibility for a chat tool, but mainly relevant for the above mentioned aspect of transmitting short messages (e.g., web addresses) during a presentation/lecture. Here it is important to

note, that the chat tool does not provide late join mechanisms. As a result one only gets the messages received during ones participation in the session and while the chat is active, not those send before one did join the chat. Such late join mechanisms are typically used by whiteboards. In principal the support for late join could be provided by the framework, but this is currently not covered by MACS.

4.2.2 Scalability issues for chat

Considering the typical short messages being transmitted, and the absence of a late join mechanism, there are no scalability problems for the chat tool. The total amount of data transmitted/received during a session by the chat tool is likely to be some hundreds or thousands of bytes. This is much less than audio or video may cause each second.

4.2.3 Structure of the chat application module

The chat tool directly follows the standard pattern for integration into MACS, as shown in Figure 15. Its internal structure is fairly simple and only consists of a functional block handling the floor control and one handling the GUI, as shown in Figure 37. As the screen space required by the chat tool is rather small it can directly be integrated into the control via the supplied panel. Assuming a rather dynamic nature of the chat channel, that is past messages are of little importance it is sufficient to just display received messages. This is somewhat usage dependent, if the chat channel is only used, for example, to pass web addresses with additional information for a lecture, past messages become a more interesting issue. Here a way to identify the point in time were the message was posted becomes relevant. This is needed to find the content matching this additional information. For the simple use assumed for this version of the chat tool a late join mechanisms, that is a mechanism able to request messages send before the time the local chat instance was started, is not required. Messages are directly transmitted via the network to all other participants, as soon as the user concludes the current line of input.

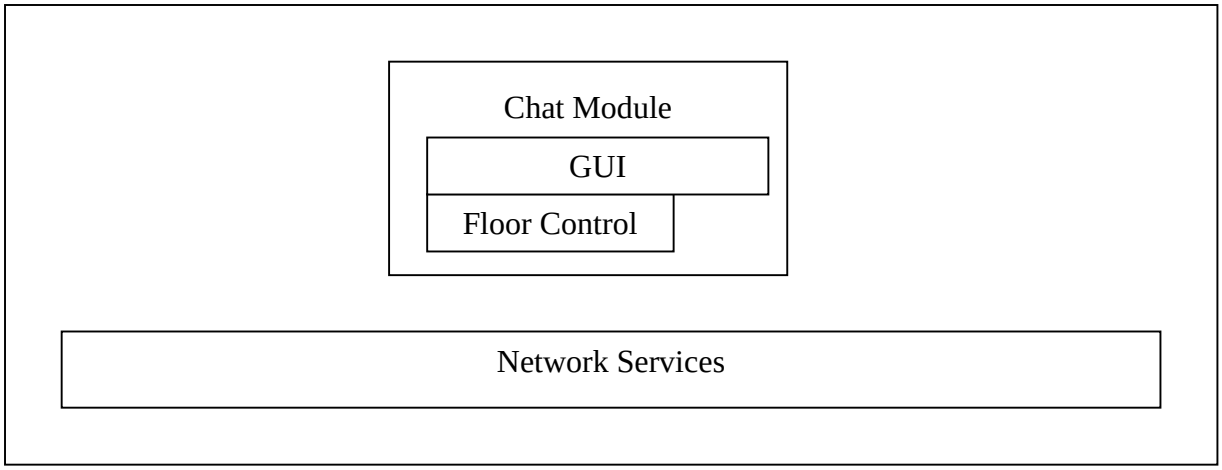


Figure 37: Structure of chat tool

4.2.4 Evaluation of the chat design

The current chat tool only provides a simple text exchange functionality and does not feature late join mechanisms, synchronization with other data streams or selective transmissions of

text only to certain group members. It does provide floor control and demonstrates basic functionality of applications within the framework.

4.3 The whiteboard application module

The whiteboard serves as a substitute for real world black boards, overhead/slide projectors and flip charts. The typical functionality of these is combined with computer related features, such as, for example, found in drawing programs. The following subsections discuss the MACS whiteboard, starting with a functional description, followed by scalability issues. Then the structure of the whiteboard itself and especially its integration into the MACS framework are discussed, before finally an evaluation of its design is given.

4.3.1 Whiteboard functionality

The design of a whiteboard has to take in account a number of issues. Objects in a whiteboard have to be reliably distributed. It is not acceptable that any object gets lost during transmission. Further the drawing order, which for overlapping objects determines which object is drawn on top must be deterministic, even more, it must be the same drawing order for all participants. For participants joining the session or at least the whiteboard media stream during the duration of the session (i.e., late join), previously send objects have to be resent, as they can still be part of the visible page and therefore are required to present a consistent view for all participants. For a tool allowing simultaneous work (i.e., in this case drawing) by a number of participants it is necessary to provide the feedback about the others actions. It is rather irritating to see sudden changes without seeing how they come about. A better approach is to show the others person cursor, as well as to show how he places or modifies objects, thus providing a better feedback. This will result in a number of additional messages being send and will place additional load on the system (including network). Further, support for tele-marker and tele-pointer helps to provide a better feedback to the user, as these allow to mark a current section of interest or allow to follow the mouse movements of the presenter.

Especially in a lecture scenario it is advantageous to employ floor control on (some of) the objects in the whiteboard. For example, it is normally not wanted that a student can delete or modify the slides of the lecturer. On the other hand it is wanted, that he can add remarks during a discussion (just like on an overhead projector). A good way to implement this is to place a restrictive floor policy on the slides, while giving the drawing floor to the current speaker (i.e., the student asking a question or explaining an issue). This requires a floor control mechanism which regulates the access to objects. Additionally, the whiteboard must be able to identify the object owner or at least the person holding the floor of an object. Only if this is the case a request to gain the floor (e.g., to ask a question) becomes possible. Providing the floor holder/owner data to the whiteboard user will in certain cases be useful and should be supported.

The above listed features and functionality are summarized below:

- substitute for
  - blackboard
  - overhead projector
  - slide projector
  - flip chart
- provide for
  - drawing capabilities
  - feedback of other users actions
  - tele-pointer/marker
  - floor control

4.3.2 Scalability issues for the whiteboard

For the use of the whiteboard in a collaborative session, it is important that all actions of a user are almost instantly visible to the other users. This is done by propagating every object creation or modification as well as every action event of one whiteboard to all other whiteboards. For example, if one user draws a line all other users are able to follow the drawing process on their whiteboard. The communication overhead imposed through the propagation of every action event is significantly reduced if the whiteboard uses a multicast implementation for its network communication. Clearly it will still not be possible to serve a very large number of participants, as, for example, envisaged in the podium discussion scenario. The in section 3.6 introduced concept of hierarchy levels could help greatly to increase scalability as the reduction of active participants makes synchronization easier, especially considering the issue of drawing order. If a passive participant gets any object with a delay it will still be displayed correctly, thus, the worst case is a temporary inconsistency between the different views. For an active participant on the other hand a conflict could arise, as he might, for example, try to modify a no longer existing object. For example, exactly the object for which in this example the delete message was lost, thus a reliable message distribution for active participants is necessary.

4.3.3 Structural and functional design issues

Integration into MACS follows the standard approach taken in the framework and is shown for the case of the whiteboard in Figure 38. The internal structure can be divided into three blocks being the graphics engine, the database and the communication services, as well shown in the same figure. A more detailed view is given by Figure 39. First the general structure is discussed, followed by a more detailed view of the functional behavior of the whiteboard.

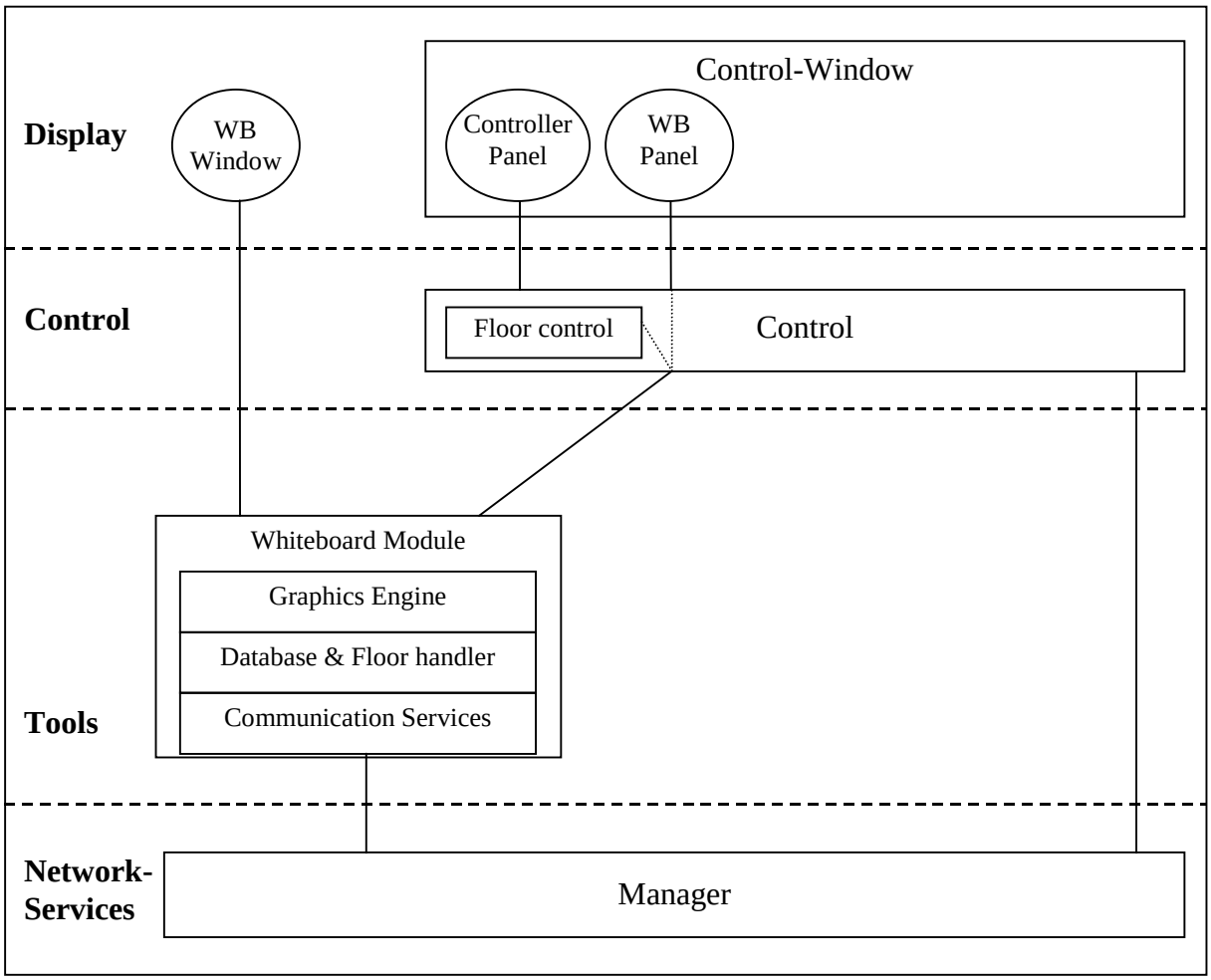


Figure 38: Basic Whiteboard structure and integration into MACS

Considering the division into the three above mentioned parts the following applies. The graphics engine of the whiteboard is responsible for rendering the drawing primitives and the slide information for the display level. The database contains these information and serves for the graphics engine as a look-up to determine which objects should be drawn (e.g., limitation by current visible page). The communication services is used to pass the user actions, additions and modifications of the current content on to the other participants in the session. Vice versa, it is as well responsible for receiving the actions, additions and modifications of the other users in the session and to pass these on to the database and thus as well to the graphics engine. The communication services handle as well the issue of late joins. If a late join occurs the communication services will send all locally created objects and objects for which the responsibility was assumed (due to the creator leaving the session) to the new participant using the later described algorithm. The aim hereby is to achieve a high degree of reliability while minimizing the network requirements. This is important, as the late join of a user will cause some peak network traffic due to the transmission of previously covered information needed to catch up in the current session. Further, considering a typical scenario with a number of people joining somewhat later as the majority, this case becomes even more important, as the number of instances needing previously send information increases.

A further point to note is that the whiteboard currently only supports the import of pictures in accordance with the in Java integrated filters. This has been extended to postscript by using an external converter (i.e., ghostscript). While this provides some problems with portability (though ghostscript is available for most platforms) it is often only of relevance to the lecturer

putting slides into the whiteboard. Further is the inclusion of postscript support in later Java versions quite possible.

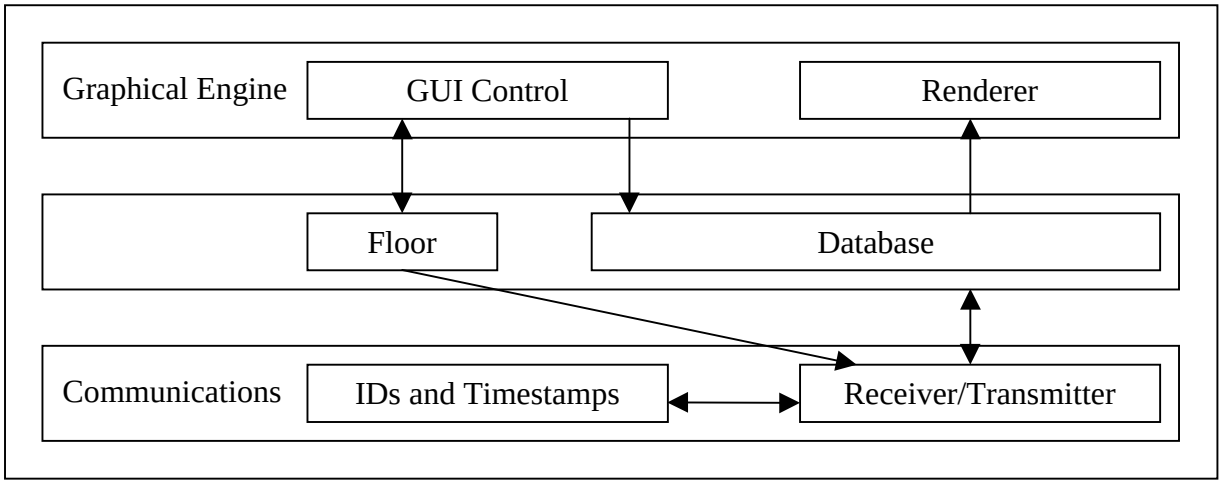


Figure 39: Detailed structure of Whiteboard

Having discussed above the basic structural division and the related functionality it is important to take a closer look on the involved details. Especially the mechanisms for late join and object hand over, as used to ensure consistency even for objects created by no longer present users, are of great importance.

If a user leaves the session, or at least closes his whiteboard during the session, another whiteboard has to take control over the objects created by him. This new owner is determined by searching for the user with the lowest ID. Currently this user is forced to take over control of the objects. A better configuration would allow to enable or disable for certain users object hand over according to their usage profile (i.e., will they be the whole time in the session or are they just having a short look) and available resources (e.g., bandwidth and available memory). Different hand over strategies will show different performance characteristics and, thus, should as well be configurable. Choosing the user with the lowest ID will lead to a concentration of objects managed by this user. This can easily be avoided by selecting a different (random) user each time a object hand over occurs. Each object and each user has an ID. The user ID is again formed by combining the local host address, the user login name and a timestamp. The object ID on the other hand is simply the timestamp of the object creation. Though this approach does not guarantee unique object IDs it makes equal object IDs highly unlikely, as the timestamp uses a msec resolution. Further, a sequence number is attached to the object counting its creation sequence for the local user. In the very unlikely case that the creation time of two objects created in two different whiteboards are identical the sequence numbers are used to determine the order. If these are as well equal (again very unlikely) an inconsistency in drawing order might arise, a case which will not affect other functionality, such as, for example, the object storage.

The late join of a user will cause some peak network traffic, as the new user has to be informed about all existing objects. In order to reduce this peak load and to potentially reduce the number of retransmits due to requests for the same large object, the whiteboard will only send basic description information about large objects, such as slides or pictures not currently displayed. These description include the position, size and a reference needed to later match the content. Only once the page is displayed the content is send and then matched to the

descriptor before being displayed. Especially for a basically sequential session, as, for example, a lecture, this will reduce peak rates drastically. Here a certain timeframe for the joining process will exist, with relatively few people arriving outside this time frame. As the participants tend to follow linearly through such a session it is likely that many of them will change to the same page at the same time (for example, based on the paging commands send by session chair) and thus the actual data for the slide or picture is only send once.

Figure 40 shows an example of three whiteboards. The first whiteboard creates some slides and thus owns these. Then the late joiner A enters the (whiteboard) session. He does consequently receive a list of all objects currently contained in the whiteboard. Here for large objects such as pictures and slides only the descriptions are send. The same process happens again if a further whiteboard joins the sessions, as B for this example. If any of the whiteboards now changes to a page containing a picture or slide it will send a page select request requesting the image data for objects on the selected page for which this data is currently not available. This data is then send by the owners. As all whiteboards receive this data they will if necessary update their entries and hence not need to request the data again once they change to this page themselves.

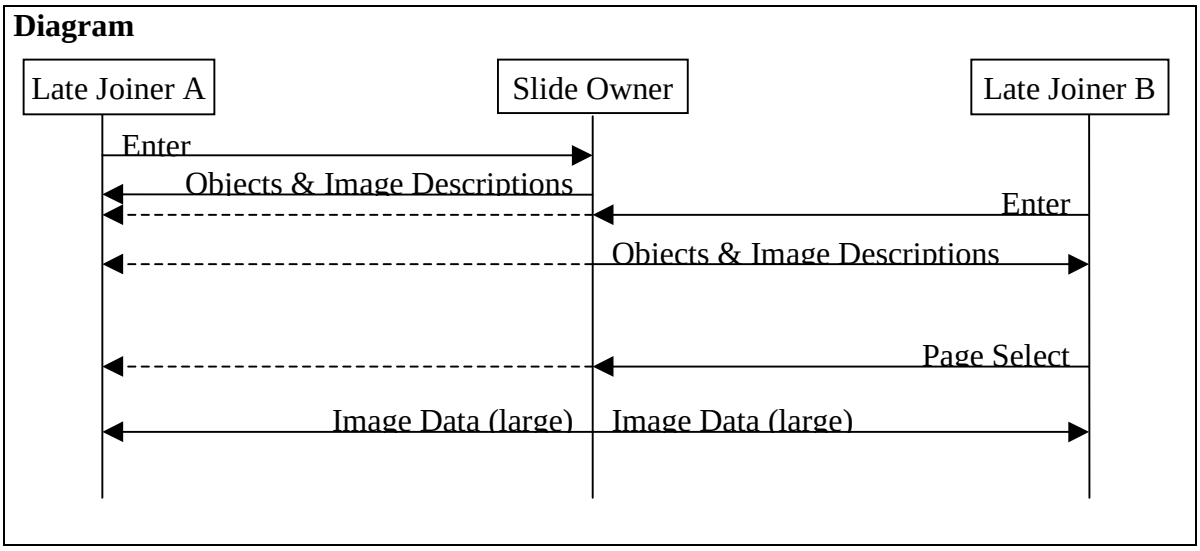


Figure 40: Mechanism to reduce whiteboard network usage peaks

4.3.4 Evaluation of the whiteboard design

The whiteboard offers a flexible and advanced functionality for presentation, drawing and discussion support. The design is somewhat resistant against transmission problems, though for normal operation a reliable network service is assumed. The customizable floor control allows to limit access to certain objects, a very useful feature especially for lectures as described before. Still the whiteboard has a number of limitations, some due to its own design and some due to its environment.

Regarding its own design it is limited in scalability as it does not allow for different hierarchy levels and as such would assume the simultaneous active usage by all participants. This is not really feasible for large groups (one image a real world blackboard with thousands of people using it at the same time). Further, two important concepts are not considered, but relevant especially in learning environments:

- private annotations
- asynchronous aspects
  - preparation of own slides
  - comparison of annotations from different people

Both these issues are outside the scope of this document, but shall be explained briefly. Private annotations are needed to attach one owns remarks to a given presentation [40]. These can be public by request or private, as they might contain secret information, as could be easily the case in a business environment. Further, these have to be managed separately from the rest of the whiteboard media, but closely integrated in function. It is imaginable to do this as a service by the framework for the applications.

Asynchronous aspects refer to the integration of the asynchronous tools in MACS or vice versa. In general one will both prepare and post process a presentation offline, that is asynchronous. MACS currently offers no access to asynchronous functionality, but for some tools, especially such as the whiteboard this would be rather helpful.

The limitations imposed by the environment are related to scalability issues in MACS or the network implementation itself and are discussed in the relevant sections.

4.4 The feedback application module

The feedback tool allows to obtain feedback from the participants, an aspect normally lost in CSCW systems due to the limited information conveyed by the computer about the other participants of the session. The following subsections will first discuss the functionality of the feedback tool and related scalability issues, followed by a description of its structure and integration into MACS. Finally some key features and limitations are discussed.

4.4.1 Feedback functionality

There are two purposes of the feedback tool, each covering one of the two corresponding feedback aspects:

- social feedback
- technical feedback

Social feedback refers to feedback about the participants in the session. Typical questions are, for example, is the presentation speed too fast, is the presentation clear or is it too difficult? In real world environments the presenter is able to determine by the facial expressions of his listeners the answers to these questions. Using a CSCW systems this will, in general, not be possible, as the video quality will be too low and normally no efficient way exists to look "from face to face" of the participants. Thus a rating mechanism is needed. This is done by providing the presenter, or his assistant, with a set of freely definable sliders. The participants of the session will be able to use these sliders in order to rate the corresponding issue and thus provide the presenter with an overall view of the opinions regarding the selected issues. Thus, one can define a slider, for example, for audio quality and each participant can submit a rating. Here each participant has the choice whether to reveal his identity or to stay anonymous. The default is anonymous operation. In this mode his rating is just added to the average normally viewed by the presenter (together with the range, that is minimum, average

and maximum values are displayed). For "objective" measures, such as audio or video quality this is of little relevance, but for more directly presenter related issues, as, for example, understandability of presentation, this is more problematic. Here the student might, for example, be afraid of his lecturer and thus would not give a negative rating if he can be identified. Figure 41 and Figure 42 show an example of the GUI for ratings, as seen by the presenter and as seen by the other participants.

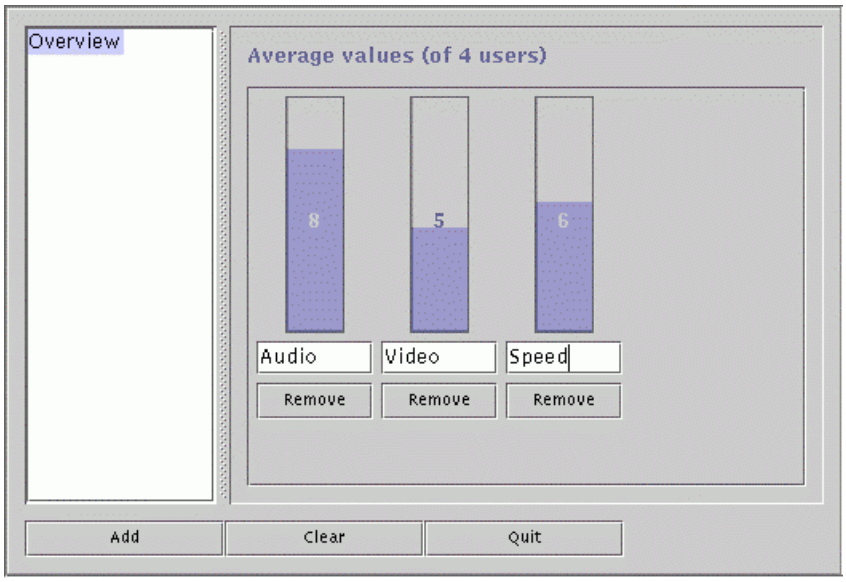


Figure 41: Feedback tool rating view for the presenter

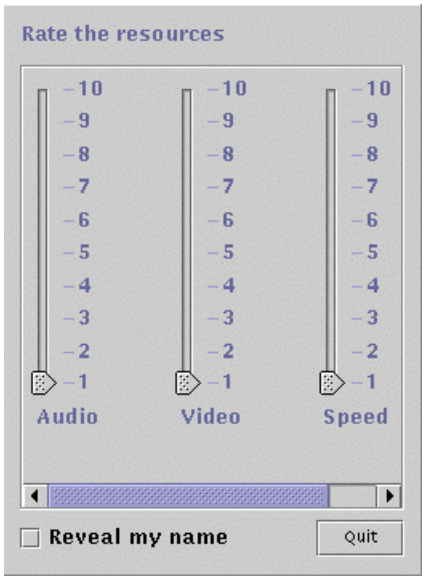


Figure 42: Feedback tool rating view for the participants

The technical aspect is much more far reaching. The simpler aspect of it is the same type of rating as described above, but with typical questions regarding audio and video quality, network delay or similar aspects. This is sufficient for the presenter. In a lecture scenario on

the other hand it is assumed that a technical assistant will be available to deal with technical problems. In order to determine if the problems experienced by a certain participant are caused by the network, his system or configuration it is necessary to allow the technical assistant to extract low level application statistics from the persons machine. A view at the packet loss rate will, for example, tell if the poor video quality is due to the network, or if an other reason must exist. This kind of functionality allows a very high degree of possible support for the participants. Clearly each participant will have the free choice whether he wants to provide the requested data, an aspect which is necessary due to privacy and security considerations. In general this will not be a problem, as the session forms a closed group and the application performance statistics of a participant are not a security or privacy sensitive issue. A simple example how such a technical view might look like is shown in Figure 43 for chat tool activity. While this is for the chat tool not really of relevance it shows the direction of development taken and will for the audio and video application modules provide a more useful insight for the technical support person.

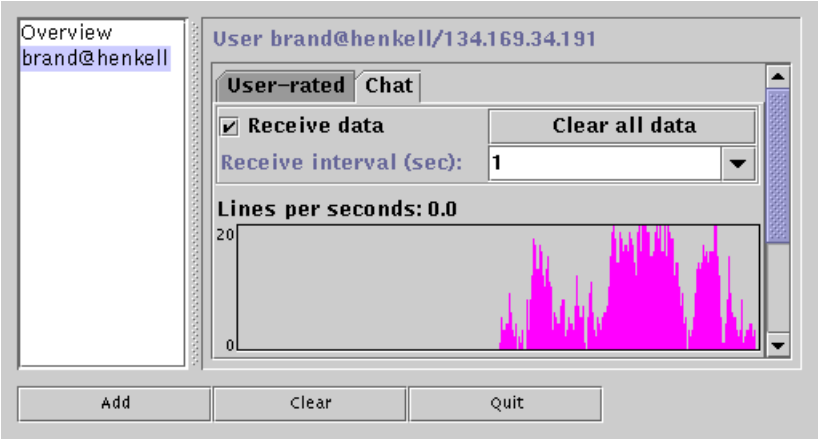


Figure 43: Feedback tool technical feedback view

4.4.2 Scalability of feedback application module

The feedback tool is mainly relevant for lecture scenarios. It explicitly tracks the ratings of each participant and thus is limited in scalability. Still it should easily be able to cope with typical lecture scenarios. Extending its usage to larger scenarios as, for example, the podium discussion an adaptation of the concept seams to be necessary. For such a large scenario a support team will become necessary, a feature currently not supported by the feedback tool. Again the integration of a hierarchical approach would increase scalability and would allow as well to extend the teaching assistant aspect to a full featured support team. Especially for a large number of ratings received as in a podium discussion a 100% reliability is not necessary, as a single missed rating will have a rather small effect on the overall rating, and thus a hierarchical concept with differences in reliability for active and passive (that is in this case passive except for supplying ratings) participants is viable.

4.4.3 Feedback application module structure

The basic design of a feedback tool is relatively easy and uses a client-server structure. This allows a high degree of flexibility. Providing an efficient screen layout for the feedback tool is a rather difficult task. Extended abilities to provide participant support by analyzing technical feedback data from remote applications provides a further challenge. The floor holder of the

feedback tool itself acts as server for all other participants, which hence are the clients. This is necessary, as it is not always of advantage to force the session creator to be the actual feedback tool supervisor (and thus server). An example is a lecture. The lecturer concentrates on the teaching aspects. A technical assistant will help in order to support the learners, i.e., the students. He will then control the feedback tool and provide support if required. The lecturer has no time to do so while teaching. He will, in general, only be interested in the average rating and maybe in the min/max ratings given to issues like presentation speed, clarity, or the perceived difficulty of the presented topic.

The integration of the feedback tool into MACS follows again the standard approach taken in the framework, as, for example, shown in Figure 15 and is here not again explained. The internal structure is divided into a server and a client version, depending who runs the application module. The clients will directly communicate with the server to submit their rating data and, if activated, the relevant application performance statistics. The server collects all this data in an internal database and generates a graphical presentation based on the users selection. Thus, the feedback tool can be divided into a GUI block for the representation of the interface, the floor control block handling floor requests, a database responsible for collecting the information and calculating minimum, maximum and average values. Figure 44 shows the detailed structure of the feedback tool including the above identified components. Here the database is only present on the server, the clients do directly pass their ratings on to the server or display the messages received from the server without storing or processing them further. The control unit is responsible for handling and processing the database entries and the resulting GUI representations. This becomes especially important if the advanced functionality of application feedback is enabled. In this case it is not sufficient to just store the ratings in the database and to form an average. In this case it is as well necessary to process the incoming application details. These have to be analyzed and prepared for interpretation by the support person. Taking audio transmission as an example, it is not sufficient to know the average number of packets arriving, neither to add the minimum and maximum values to this information. A measure for the jitter and for the variations in number of packets arriving is more useful. Thus, the incoming data has to be processed and represented in an adequate form, as, for example, a graph.

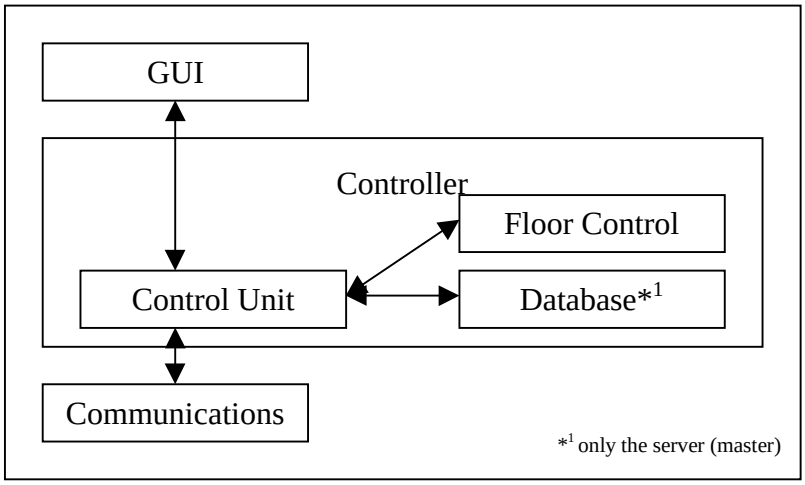


Figure 44: Details of Feedback tool internals

Especially for the technical feedback a flexible adaptation of the feedback tool is important. To obtain the application specific data a dedicated interface exist. This is queried by the

feedback tool for the possibly available data. For each possible data the type and name are identified. The feedback tool can now query this data and, thus, retrieve the values. It is important to note that the data is not interpreted, thus flexibility is maintained by allowing later integration of new applications. The provisioning of this technical feedback data is up to the specific applications and is in design and implementation independent of the feedback tool itself. The feedback tool has no hard-coded technical feedback panels/tabs to display the retrieved information, but generates these dynamically based on the information extracted during setup, as described above. An example for such information is:

- audio receiver module
  - "packet loss" type integer
  - "network status" type string
  - "data rate" type float

These information are parsed and a panel/tab is constructed to display this information, as shown in Figure 43. This allows to add new applications to the system and gain technical feedback from them without needing to adapt or recompile the feedback module in any way.

4.4.4 Evaluation of feedback design

The scalability of the current design is somewhat limited, as explained above. For the envisaged usage scenarios this is not an important aspect, as these can be efficiently served. While for the normal lecture scenario the support by a single teaching assistant should be sufficient, some scenarios exist were an extension of this concept to a whole support team would be advantageous, as has as well been described above. These are only small limitations for a very flexible support tool as the feedback tool. Especially the detailed feedback allowing support during technical or configuration related problems is a great step forward to a more complete and advanced system usable for the next generation of tele-teaching and groupware applications. Some usability questions remain open, but these have to be evaluated during more excessive field tests by experts in the areas of didactics and psychology and are outside the scope of this work.

5 Interworking with other CSCW systems

Interworking capabilities allow one type of system to work together (i.e., interwork) with a different one. This is very useful due to a number of reasons. Different companies, or even different sections in large companies, will have heterogeneous IT infrastructures and will thus not all use the same CSCW system. As a result they can not easily work together as the systems are, in general, incompatible. The introductory scenario is a good example. Today large automotive companies spend enormous amounts of money on trying to establish a company wide IT infrastructure and well defined interfaces to the suppliers. Often these are actually required to adapt a certain, by the large automotive company prescribed IT infrastructure themselves. While this is of advantage to the automotive companies, it may be problematic for the suppliers as the system they have to employ is not well suited to their needs. One reason can be that the system is not suitable for the small size of the supplier himself. The usage of a system allowing interworking would prevent this type of conflict.

A number of important points have to be considered when discussing interworking. It is not always possible to achieve a good integration and therefore far reaching interworking capabilities. Each system has its own very specific features. These can not always be transformed in order to work together with the specific features of the target system (i.e., the system the interworking is wanted with). This is especially true for the tools of the CSCW systems. Establishing a joined session, that is allowing the systems to connect to each other and to integrate the other user into a session, is only the basis to actually start to interact. The actual interaction is done via the tools, which need to be compatible as well. Tools using standard, CSCW system independent, protocols for coding their media are much easier to interconnect then those using proprietary protocols. As a result audio and video tools are normally much easier to adapt, than, for example, a whiteboard. The whiteboard on the other hand still uses fairly simple and often between systems very similar features and therefore is again easier to handle than highly complex tools which might represent their data in a totally incompatible way (e.g., CAD tools).

First some basic interworking issues are discussed, followed by the aims of the interworking in MACS. Then for each of the two sample systems selected a detailed analysis is presented and the design of a MACS interworking module discussed.

5.1 Basics interworking issues

A steadily growing number of CSCW systems is being used today despite their current limitations. Replacing an existing system will cause costs related to purchase, installation and instruction of users. Even more complicated is a situation which includes different companies or, in big companies, different sections. Here it might not be possible to reach an agreement on using only a single system (probably these will be political reasons and not technical reasons). These situations require systems to be interoperable, that is to support the interworking of different systems. The advantages of interworking are fairly clear. No new installations or instruction of users are required. The disadvantages are quite obvious too. For two different systems to interwork they have to reduce their functionality to a common subset. As a result most of the advanced features will be disabled for at least some participants. Actually it is probably preferable if they are disabled for all, as otherwise some participants might miss information without even noticing it. This is not only true for session management, that is locating and joining a session, but especially for the tools used in the session. For audio and video tools it is, in general, easier to achieve interworking than for data tools. This is due to the adoption of certain audio and video coding standards by a wide

variety of such tools. The data tools on the other hand tend to implement their own protocols and data structures. A whiteboard is a good example of such a case.

To enable interworking of two systems the following lists the main issues to consider:

- basic system communication
  - compatible network infrastructure and protocols
- session setup/tear-down
  - session and user information exchange
  - capability exchange
  - create session
- session conduction
  - session control (join, invite, leave, expel and others)
  - floor control
  - application control
    - starting application
    - capability exchange
    - compatible media (streams)

Overall is interworking an important aspect of a CSCW system and might be of fundamental importance regarding its acceptance/deployment. Currently the two most important standards are the ITU-H.323/T.120 standard family and the standards established or being developed for the MBone.

The MBone is a virtual overlay network covering parts of the Internet. It allows the transmission of multicast data within this network and therefore provides an efficient support for broad-/multicasting information. The support for basic interaction with systems developed for these standards should be considered in the design of any new CSCW system, despite the increasing number of custom extensions and variations existing for both the ITU standards and the MBone. On the other hand one has to consider, that in a commercial environment interworking support is normally limited and often not really wanted. After all the companies producing CSCW systems want to sell their system and not enable the usage of someone else's system. As a result interworking is done, if at all, by supporting to some degree open standards, as the ITU-H.323/T.120 family. Often these standards allow for extensions and these are commonly used, resulting again in incompatible (advanced) features.

The two most important systems using CSCW relevant standards are currently systems based on the ITU H.323/T.120 protocol families and systems designed for the MBone. As a result two interworking modules were designed for MACS, allowing to interwork with ITU and MBone based systems. Interworking modules are quite complex, more so than most other modules. They need to access a wide variety of important system resources, especially the control databases. In general these modules will use their own thread for bookkeeping. Each interworking module must at least implement the following functionality:

- user discovery (i.e., how do I find the information needed to contact the wanted user),
- session join
- session leave

Optionally further control functions like invite and expel, various floor control functions and even state visualization functions might be implemented. The modules are mainly responsible for the interworking of the systems control functionality, especially the session setup and conduction aspects. The interworking aspects of the applications normally have to be handled differently, that is by the applications themselves or by a further adapter module.

5.1.1 ITU-H.323 and T.120 families of standards

The ITU-H.323 standard family [54] describes equipment and services for packet based multimedia communication over networks without QoS support. A good introduction of its use in the typical Internet environment is given in [96]. [31] discusses its use in LAN and ATM environments. The ITU-T.120 protocol family [55] covers the different aspects of support for multimedia conferencing from a more application data oriented approach. Looking at a typical system one could roughly split it between the two standard families [87] by saying, that the audio and video communication is done via the H.323 standard family, while data communication such as chat, whiteboard [60] and file transfer [61] is done via the T.120 standard family. Both originated in the telecommunication sector and were then adapted for the use in typical computer networks.

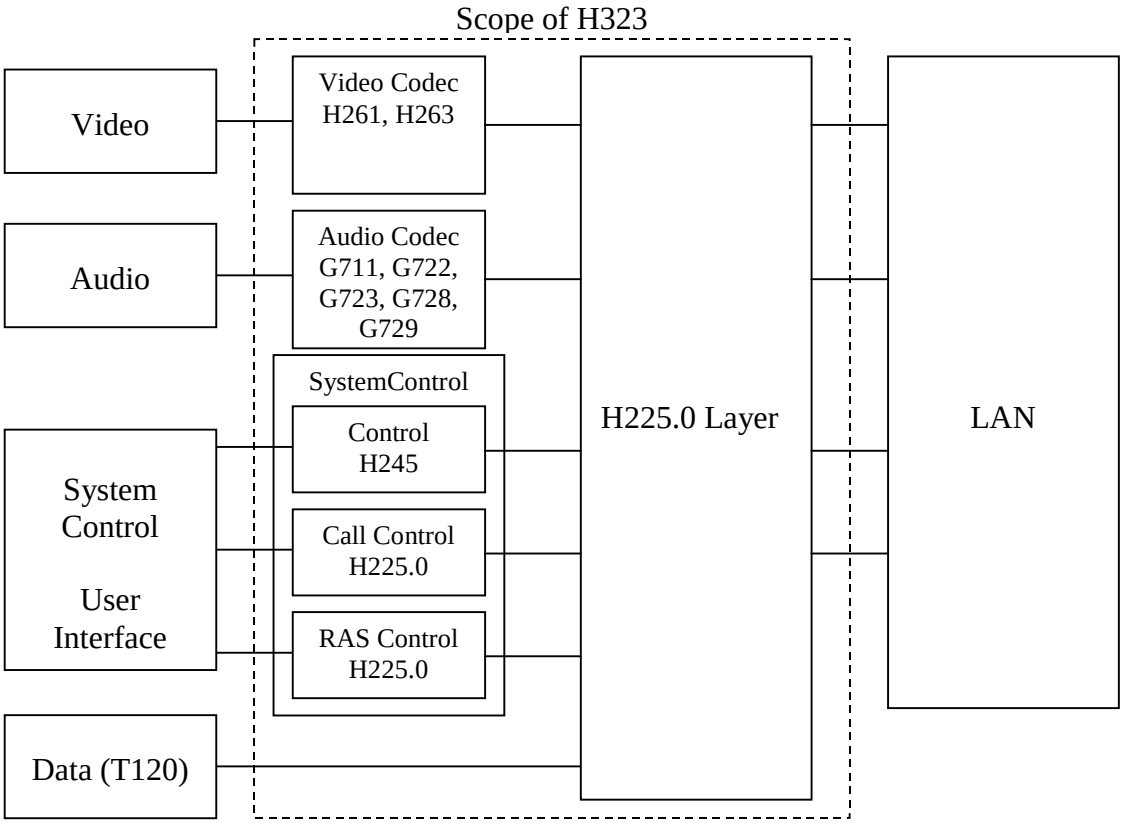


Figure 45: Scope of ITU H.323

The scope of the H.323 covers most of the system, only excluding the LAN, Media equipment (Audio, Video) and the application itself with its control functionality and user interface. The H.245 [53] is used for control messages. The H.225.0 [52] describes the underlying protocols used for media packetization and control in the H.323 system, while RTP (Real Time Protocol) [27] is used for the actual media packetization.

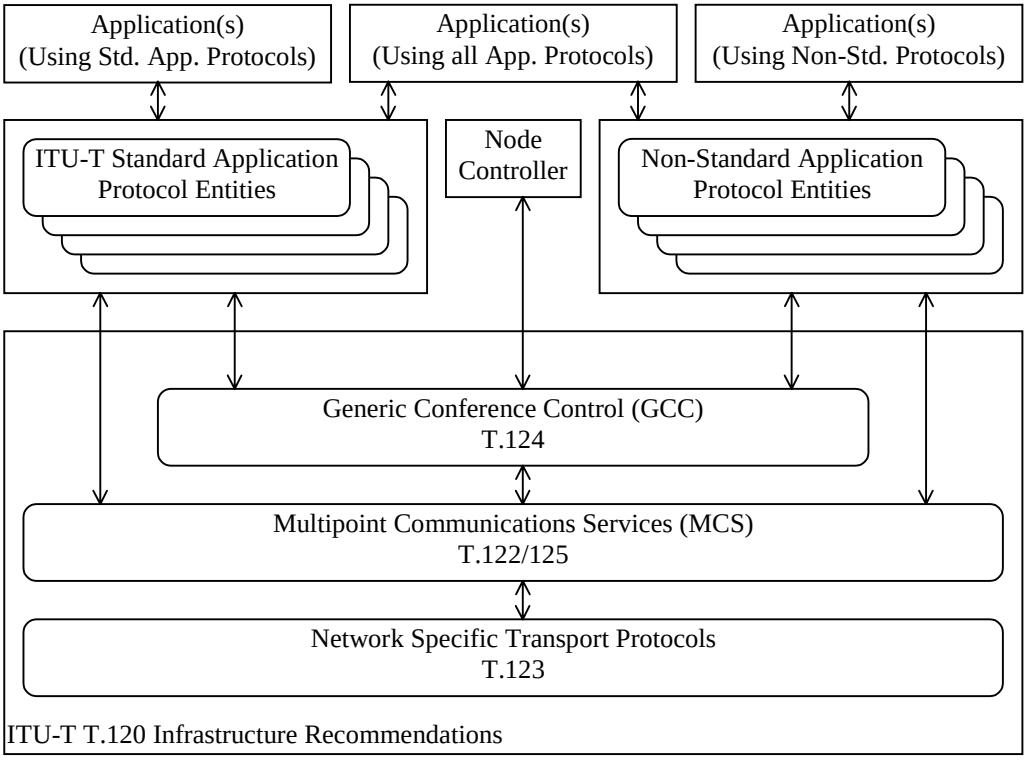


Figure 46: T.120 standard structure

The T.120 covers most of the data communication part of the whole system. Starting with the description of network specific transport protocols by the T.123. These are used by the Multipoint Communication Services (MCS), that is the T.122 [57] and T.125 [59] to define a channel based communication structure. The Generic Conference Control (GCC) as defined by the T.124 [58] is responsible for managing the conference relevant data and provides conference control mechanisms. Applications can use standard protocol entities, such as the T.126 for a whiteboard, or define their own custom, non standard protocols based on a generic template provided by the T.121 [56].

The advantages of the ITU-H.323 and T.120 standard families are their well developed state, the large number of existing systems, its versatility and company independent development (commercial dependencies are therefore not given and systems of different companies should be interoperable). The disadvantages are that these standard families are huge and do not scale well. The resources, both in man power to develop and implement and in memory and processing power to execute are often unacceptable. Due to its origin in the telecommunication field some concepts do not scale well for use on modern computer networks (e.g., regarding support for multicast networks).

5.1.2 The MBone

The MBone is a virtual overlay network on top of the Internet. Multicast enabled areas are connected via tunnels. The MBone covers wide areas of the Internet. Data is transmitted in multicast-datagrams within multicast-groups.

To support multimedia content and conferencing a number of protocols have been developed. These are not necessarily "MBone" protocols, but have been developed considering their use in the MBone, or been extended for that purpose. In some cases they have simply been reused as they were suitable.

Taking the example of a typical audio/video session in the MBone. A description of the session is created. The Session Description Protocol (SDP) [45] is used for that purpose. This session description can then be made available to others through a number of ways. It can be send by email using the Session Initiation Protocol (SIP) [46] or directly via announcements send on the MBone itself, packed according to the Session Announcement Protocol (SAP) [44]. In the session description the available media streams, in this example audio and video are described. This will include all information necessary to receive these including their coding and protocols used for transmission (typically RTP [27]).

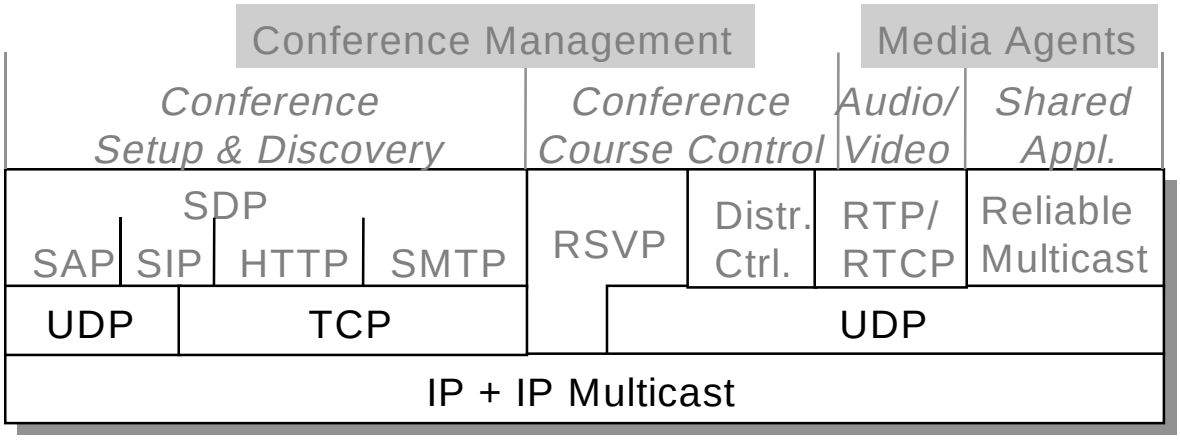


Figure 47: Overview of typical standards used in the MBone

The advantages of MBone generally are the high scalability based on the efficient data transmission and, therefore, relatively small bandwidth requirements. The disadvantages are the lack of detailed control mechanisms and some security issues.

The following sections will first discuss the interworking aims of MACS, followed by a section identifying two target systems. For each of these the integration in MACS and its interworking capabilities are discussed.

5.2 Aims of interworking in MACS

MACS aims at supporting interworking with some standard CSCW systems. This is achieved in a flexible manner by using a modular approach. For each target system for which interworking capabilities are desired an interworking module is designed and implemented. This illustrates well the flexibility of the modular approach in MACS. A further advantage of

this modular approach is the better resource usage. In the case of the interworking modules these are only loaded and activated on demand.

The aim of MACS is to allow at least to establish a joined session with these systems. Further an integration of the media tools, that is audio and video is envisaged. The integration of other tools is discussed in the following sections but considered of a lesser importance as the main point is to provide a prove of concept.

5.3 Interworking with ITU-H.323/T.120 based systems

Quite some ITU-H.323/T.120 compatible systems exist, or at least systems which claim to be compatible and implement the most basic functionality of these standard families. For comparisons it is useful to select a preferably widely deployed representative for the ITU-H.323/T.120 based systems. This will be done next, followed by a discussion of the requirements needed for interworking with it. Then the design goals for the MACS interworking are stated and the structure of the interworking module shown. Next the differences between MACS and the ITU protocols are analyzed and solutions for mapping the functionality are discussed. Finally an evaluation of the interworking capabilities with ITU-H.323/T.120 based systems is given.

5.3.1 NetMeeting in more detail

An attractive representative of the category of ITU based systems is Microsoft's NetMeeting[3]. It provides an easy to use interface, as shown in Figure 48, for establishing audio connections via the Internet. Both in symbols and terms it is clearly orientated towards telephony, making it easy for the user to learn the basic operation. Further, video support is provided. It is possible to receive the video of the person one speaks to and to send ones own video. To see ones own video an additional window has to be opened. A whiteboard and a simple chat application are provided, as well shown in Figure 48.

The whiteboard offers a fairly complete and comprehensive set of tools, sufficient for any small meeting. Two further, probably commonly not as much used tools are the file transfer and the application sharing. While the file transfer is very convenient, the lack of control over the received files is somewhat disturbing.

The application sharing features two levels of control, the ability to show the other user any application window and to let him work in this window. While for simple tasks this provides many powerful possibilities it is not really possible to use this in a CSCW context. This has two basic reasons:

- inflexibility
- security

Inflexibility refers to the lack of local adaptations. The application window shared will be shown to all users at the same size and location as the original. This will be very inconvenient to some, as they will have an arrangement of their windows not suitable for that location and size of the shared window. Windows covered by the shared window are not easily accessible, as input in this area will be directed to the window owner. Further, in certain cases of different

[3] Version 3.0 of NetMeeting was used. It can be obtained from http://www.microsoft.com/netmeeting.

screen size it is even not guaranteed, that the other person will be able to see the (important) parts of the shared window at all.

The second issue are the security problems. Once the person sharing the window allows others to work in it, he looses control. While he can regain control at any time, he has to carefully monitor the other user, as he can do any operation permitted by the shared program (e.g., if you share the Explorer, you can format the other persons hard drive). As only one person has the input focus at any point in time a true simultaneous (collaborative) work is not supported.

This leads to a further important point to note, that is, NetMeeting is basically designed to work as a two persons video conferencing tool. While it is possible to use it with more participants this is not easily done. For audio and video special additional hardware (a MCU) is needed. Using a powerful host the MCU capabilities can as well be implemented in software and some implementations for this do exist. Both hardware and software implementations are very expensive.

Still, overall one can say that the user interface of NetMeeting is as well organized and mainly aimed at sessions with only two participants. It is not possible to see more that one video feed at a time and the feedback about the additional persons is only provided via a (small) list of users.

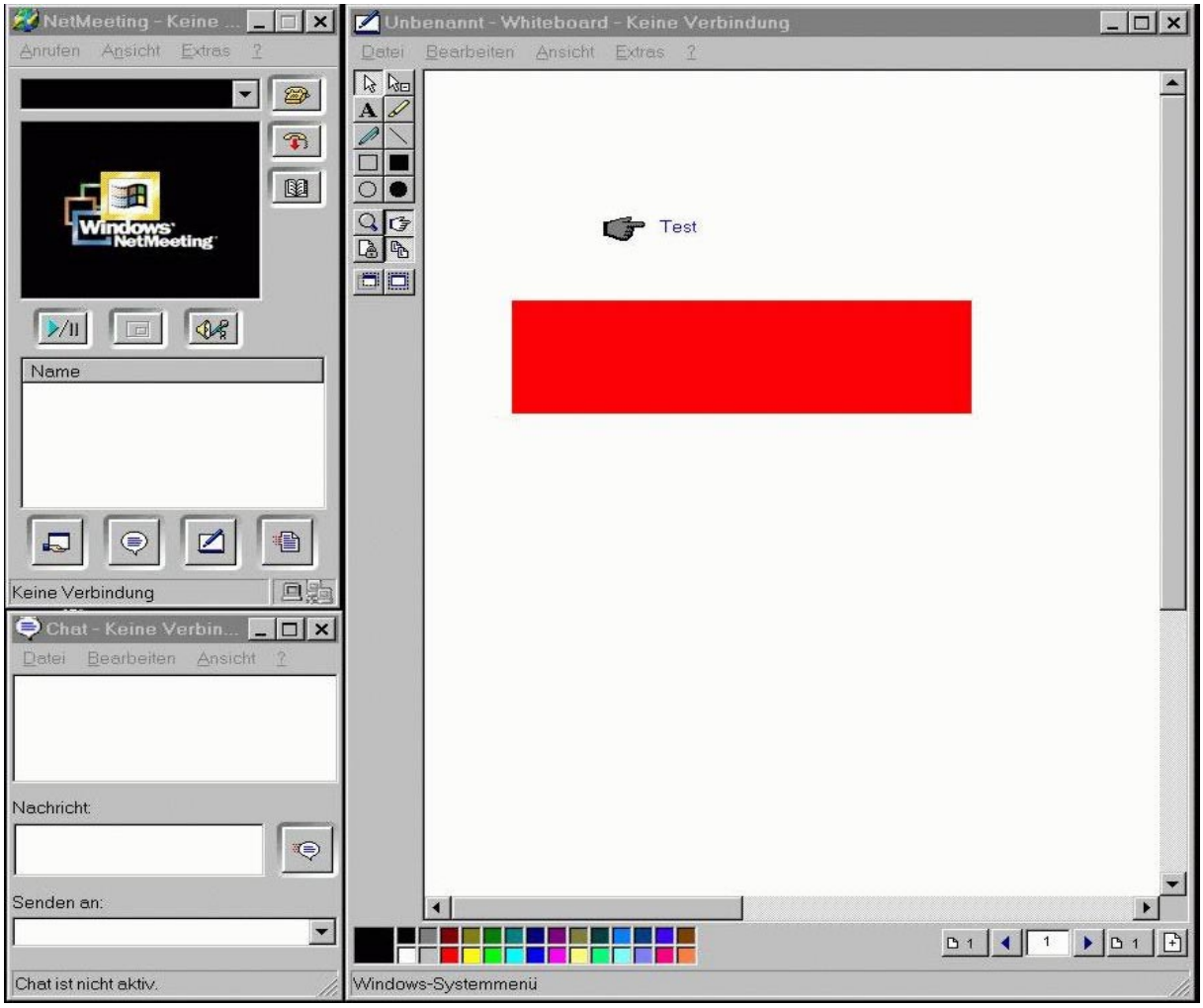


Figure 48: NetMeeting (Control, Chat and whiteboard)

5.3.2 Requirements for interworking with NetMeeting

The aim is to provide a basic interworking support between MACS and NetMeeting. In order to achieve this the following requirements exist:

- establish connection between MACS and NetMeeting
- integrate session setup, conduction and tear down
- setup compatible tools

These are the minimal requirements. The two systems must be able to communicate and follow the same protocol to setup a session, that is one creates the session and the other joins or is invited. Once both participants are in the same session compatible tools have to be set up to directly communicate with each other, thus, allowing the participants to interact. To support incompatible tools a media converter agent has to be implemented, as shown in Figure 49. Hereby the media converter can be a separate unit or be part of any of the two systems. In order to implement a media converter the protocols and data structures used by the two incompatible (but clearly as a requirement similar) tools have to be known. Most audio and video tools use compatible coding/decoding standards and hence they are normally relatively easy to integrate into such systems. Thus, for MACS the audio tool was chosen as an example.

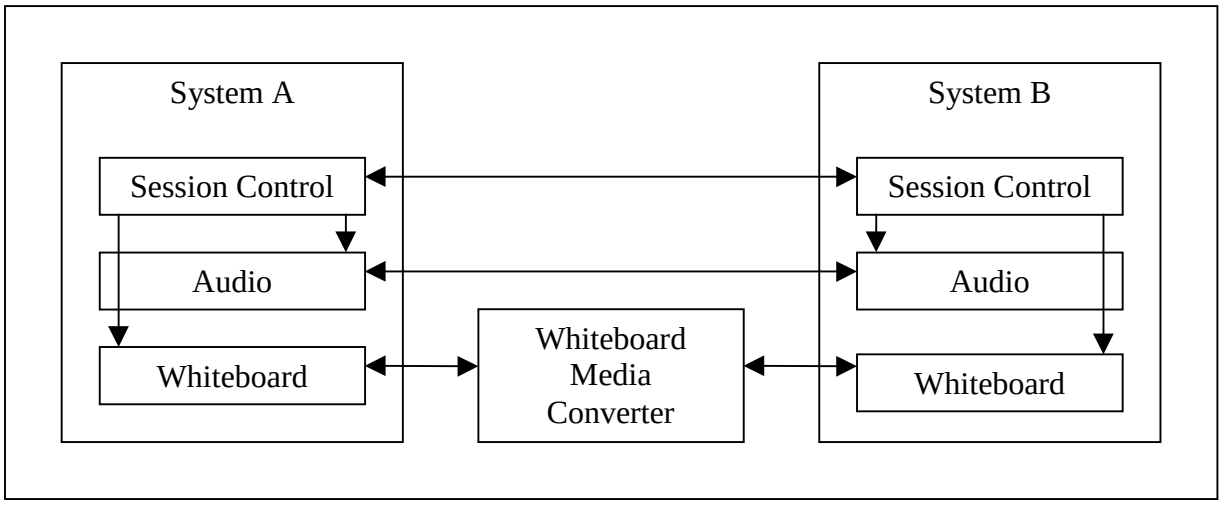


Figure 49: Interworking of two systems and media converters

5.3.3 NetMeeting interworking design goals

From these requirements and the overall aims of MACS the following design goals can be deduced. The interworking module should

- be an optional MACS component
- allow transparent integration into MACS
- provide basic functionality
  - allow session setup, conduction and tear down
  - provide one tool to demonstrate interworking

To achieve these goals a number of design issues have to be considered. The design of the framework allows for modules to be loaded into the control if specified by the current configuration. This will meet the first point, the optional integration into MACS. The interworking module will have to enter the relevant session data about sessions and users offered by NetMeeting into the corresponding MACS session and user lists. Further, the normal join and invite mechanisms have to be implemented. This will provide a transparent integration into MACS. Information about the status of the interworking module can be accessed via the module tab, as described in section 3.5.3. Therefore only the tool integration remains to be considered. The audio tool was chosen as an example. As NetMeeting and MACS both use standard coding for their audio tools the interworking between these two implementations requires only to setup a direct connection between them (i.e., assignment of a common address) and to ensure their correct initialization (i.e., to provide them with the correct information about the audio streams to generate according to given sample rate, resolution and coding)

5.3.4 Structure of ITU interworking module

Before analyzing the differences between MACS and the ITU protocols it is useful to discuss the basic structure of the MACS ITU interworking module in order as to facilitate the discussion about the solutions for mapping differences between NetMeeting and MACS.

The ITU interworking module is loaded (and hence embedded) into the control if so specified by the current configuration. It will interact (indirectly via the transaction system) with the communication services and the database, as shown in Figure 50.

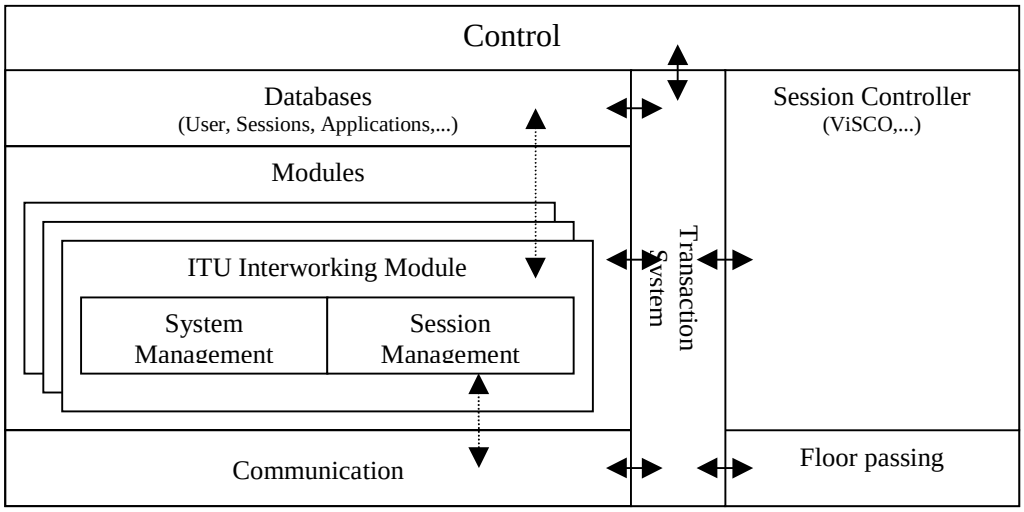


Figure 50: ITU interworking module integration in control level

The ITU interworking module has two basic functions, the system and session related management. System management refers to the basic management of incoming connections, while the session management is responsible for handling the participation of a NetMeeting user in a MACS session. Figure 51 shows a more detailed view of the internal structure of the interworking module.

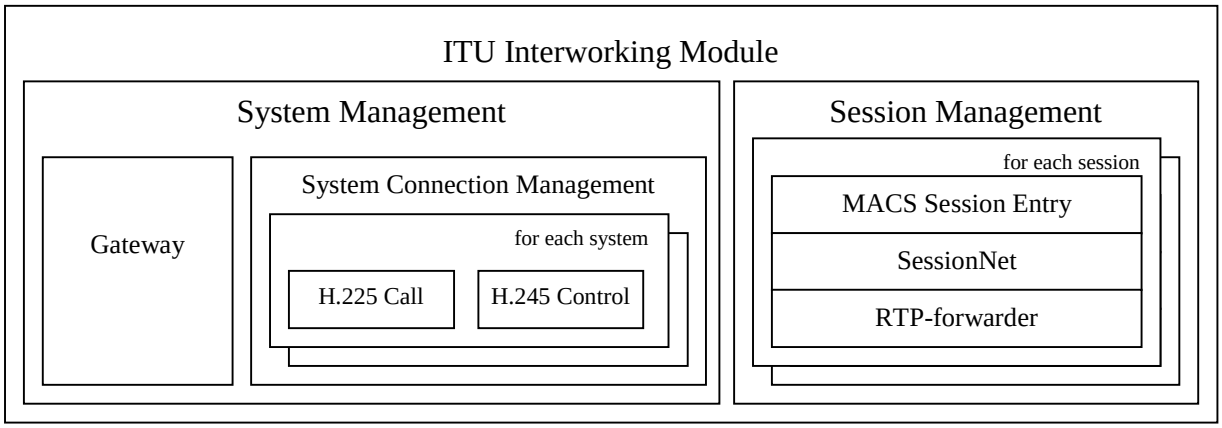


Figure 51: Internal structure of ITU Interworking Module

The gateway thread is the initial contact point for an incoming connection from a NetMeeting user. For each system contacting MACS the gateway thread creates an entry with the system connection management and establishes a connection according to H.225 and H.245. If this system (or user) is now invited into a MACS session the session management will create an entry matching this user. This includes a session entry and a corresponding SessionNet (that is a private protocol converter instance) which is directly associated with the user entry. Further, a RTP-forwarder is created (if needed) to allow the forwarding of RTP streams (i.e., audio) by MACS. This is necessary in order to transmit an incoming unicast connection to a possible multicast group, as preferably used for audio by MACS itself. More detail about the structure, protocols (see appendix A.4) and functioning, including that of the RTP-forwarder, can be found in [78].

5.3.5 Interworking between MACS and ITU-H.323/T.120

This section takes a look at some of the key differences between the ITU based system NetMeeting and MACS. The different aspects considered are:

- system/connection structure
- communication overhead

It is important to note that in a certain sense a ITU based system has two different aspects to it, which are mainly independent. The audiovisual part of a session is covered by the H.323 while the data related parts are covered by the T.120. As a result a certain overlap in functionality exists. This can be seen in Figure 45 as here the T.120 is not connected to the system control [31] and internally implements its own system and session control, as explained below.

System and connection structure

The ITU is, in general, strongly oriented towards typical telephone networks and has only adapted support for typical computer networks later on. This can often be seen in the initial setup phase of a connection, where the handling is oriented towards establishing (as in switching) a connection first, while Internet based systems such as MACS assume that the other system is already connected to the network and only needs to be contacted.

For the data part in ITU based systems the T.120 is used. Here a session is managed by a central instance, both for the MCU (T1.22/T.125) and GCC (T.124) level with the nodes organized in a tree structure. Complete copies of the configuration are kept in all nodes. A change is propagated upwards, possibly merged with other changes. The top level node will then send an update down the tree. This causes a high overhead in the processing and administration of changes and, thus, does not provide a highly scalable approach.

MACS follows partially a similar schema in that a single instance is as well the master for changes to the database. The difference here is that the master is directly contacted, not placing additional load on other nodes. This does not increase the number of requests significantly, as in the other approach all requests finally are as well routed up to the top level node. If a server group setup is active, MACS can even manage the administration more efficiently, as the work is split between servers and a number of actions can be directly carried out by a single server without involvement of the other servers. For some actions this is not the case, hence in these cases no real difference to the single server setup exists.

For interworking the by the ITU used tree structure and the by MACS used flat structure are hard to combine for the general case, as in a free mixture of both. Thus it is necessary to consider a simpler case. The simplest case is to assume MACS to be the ruling instance in the ITU tree structure. In this way a single node, or even a sub tree of ITU nodes, can connect up to MACS, as shown in Figure 52. This is the approach taken for the ITU interworking module. As a result it is possible to include an ITU node or nodes including sub trees into a MACS session.

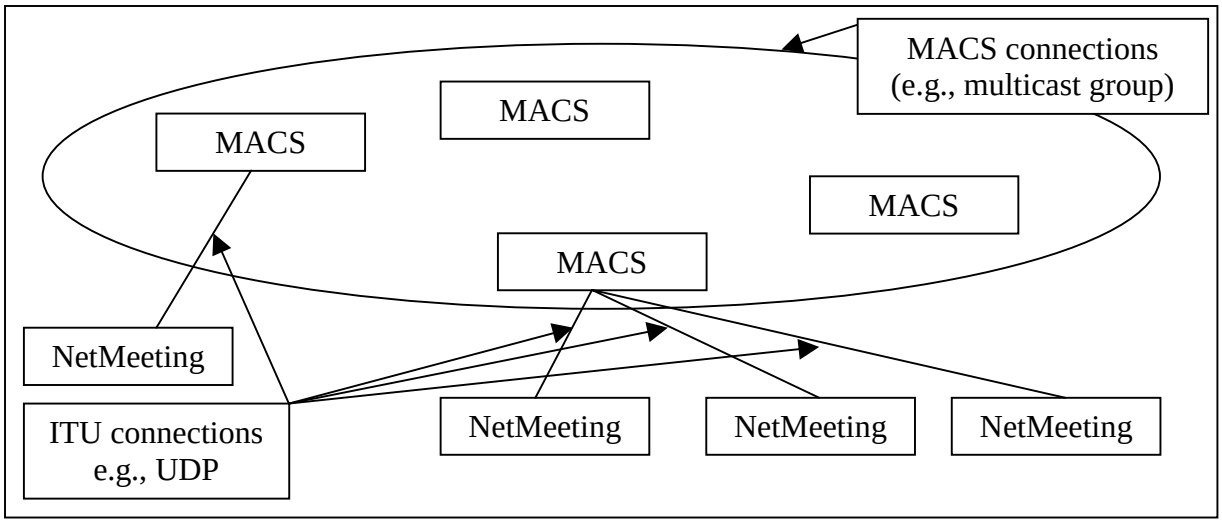


Figure 52: Connection model for interworking with ITU based systems

It is not possible to join an existing ITU session with MACS, as in this schema MACS would not be the highest order instance and this is required for the interworking module to work. The alternative, to implement MACS as a client, is possible, but has a number of drawbacks:

- reduced functionality
- highly increased implementation effort

First of all the advanced functionality of MACS will have to be disabled, as it is not supported by NetMeeting, leaving little advantages to the use of MACS in such a situation. Further, a simple NetMeeting instance without a MCU support is only in a very limited way able to handle multiple clients, thus reducing the usefulness of this approach even further.

The second point is that the implementation of MACS as a master requires less manpower. This is due to a number of reasons:

- simpler topology (star)
- simpler internal structure
- simpler resource management

Latter two are mainly a consequence of the first point, being a simpler topology. Normally a tree topology is assumed. This is a more complex topology as a simple star, were the master can directly reach each of the clients and vice versa. For MACS to operate as a node in a tree it would require to implement message forwarding including possible message merging for messages received from nodes connect to the local MACS instance (lower order nodes or leafs within the tree topology). This is not required if the MACS instance is the master and a simple star topology is enforced. Further, the resource management becomes easier, as the master contains all the relevant data to allocate resources, which in the case of the clients could require the exchange of additional messages.

As a result only the "MACS as a master" approach is used in the ITU interworking module. In order for the session setup, conduction and tear down to work the different protocol entities, that is the different messages send have to be interpreted and mapped by MACS in such a way as to emulate a higher order ITU instance. Taking the setup of a session as an example (see as well [78]) the following steps, as shown in Figure 53, are necessary in order to include a NetMeeting user in a MACS session, assuming the case were the NetMeeting user initiates the connection (for the inverse case only the Q931 messages have to be reversed).

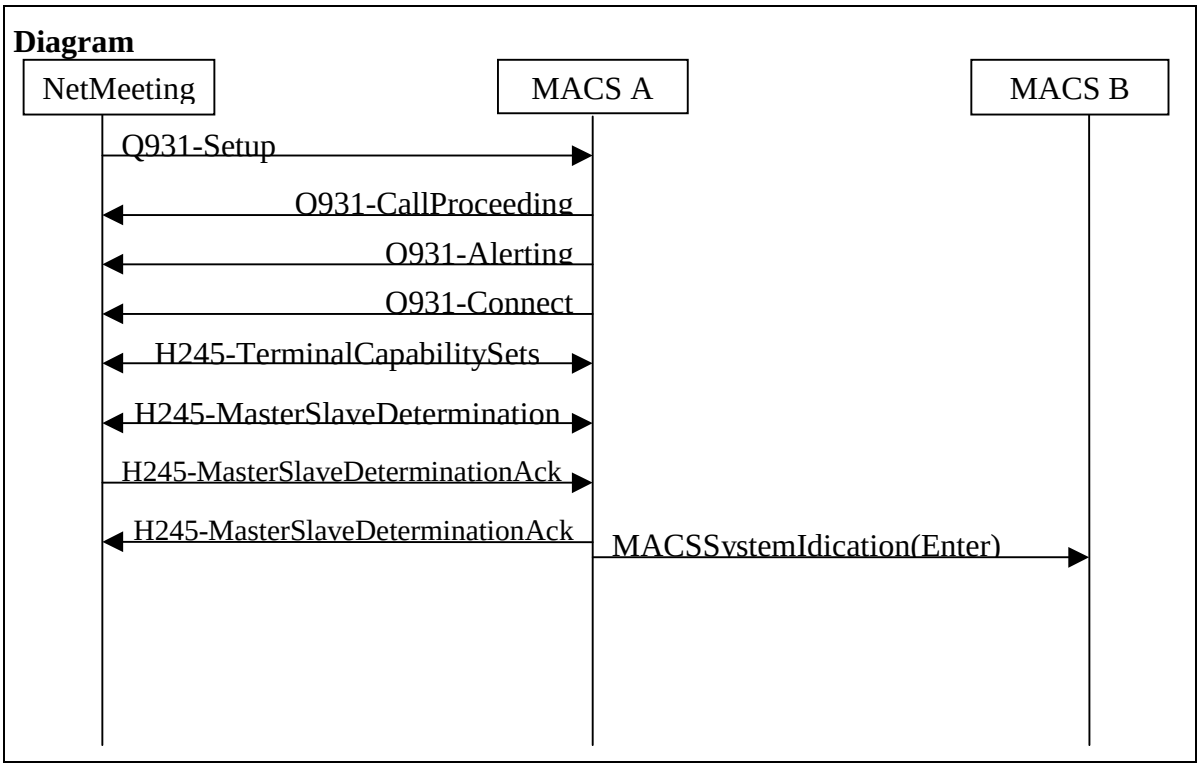


Figure 53: Message mapping between MACS and NetMeeting

First a Q.931 connection is established by NetMeeting with the ITU interworking module in MACS (in the figure MACS A) acting as a gateway in the H.323 sense. Once this step has successfully concluded a H.245 connection is established and the terminal capabilities are exchanged. Next the determination of the master is initiated by the calling side. This is normally done by generating a random number and sending it. The instance with the largest number will win the exchange and consequently become the master, while the other side will act as a slave. MACS will send the maximum value and thus gain this exchange. If the other side did as well (by design or by coincidence) send the maximum, then this process will be repeated. If again the master can not be determined the connection setup is aborted and thus fails. On success the basic connection process is completed and the MACS system (MACS A) will start sending system indications for the NetMeeting system to other MACS systems (e.g., MACS B). Basically the MACS system with the ITU interworking module (MACS A) will from now on act just as if it is not only one system, but a number of MACS systems. One for its actual user and an additional instance for each connected NetMeeting. This process is transparent to the user.

At this stage the MACS users can see the NetMeeting user in the user list and, thus, for example, invite him into a session. This will cause the association of the established connection between MACS and NetMeeting to a specific session. It is not possible for a NetMeeting user to participate in multiple MACS sessions simultaneously, a feature anyway not supported by NetMeeting. For each user in the session a H245-TerminalJoinedConference message is send to NetMeeting providing it with details about the other session participants.

The integration of the NetMeeting user into a session is now complete, but this is not yet sufficient to actually communicate with each other. To do so a tool, as, for example, the audio tool has to be started. Hereby it is the responsibility of the session control to handle signaling in order to enable communication between the two different audio tools (or instances). This includes address as well as media coding information. Here two basic setups are possible:

- direct communication
- indirect communication

In the case of direct communication the NetMeeting tools (e.g., audio) will directly communicate with the MACS tools, while in the second case the communication between NetMeeting and the rest will be routed via the MACS instance with the ITU interworking module. This might become necessary, if the NetMeeting tool does, for example, not support multicast as used by the MACS tools and, thus, needs an instance which will resent its data into the multicast group and vice versa resent the data received in the multicast group over the direct connection. Thus, the following cases for a initial application start within a session can arise:

1. NetMeeting first
   a) connected MACS is chair
   b) connected MACS is not chair
2. MACS with directly connected NetMeeting first
   a) this MACS is chair
   b) this MACS is not chair
3. other MACS first
   a) this MACS is chair
   b) this MACS is not chair

The basic startup sequence for case 1a and 2a is the same, only in the second case some additional complexity on the MACS side is required to setup this forwarding functionality. Taking a closer look at case 2a, as shown in Figure 54, the following sequence of messages can be identified.

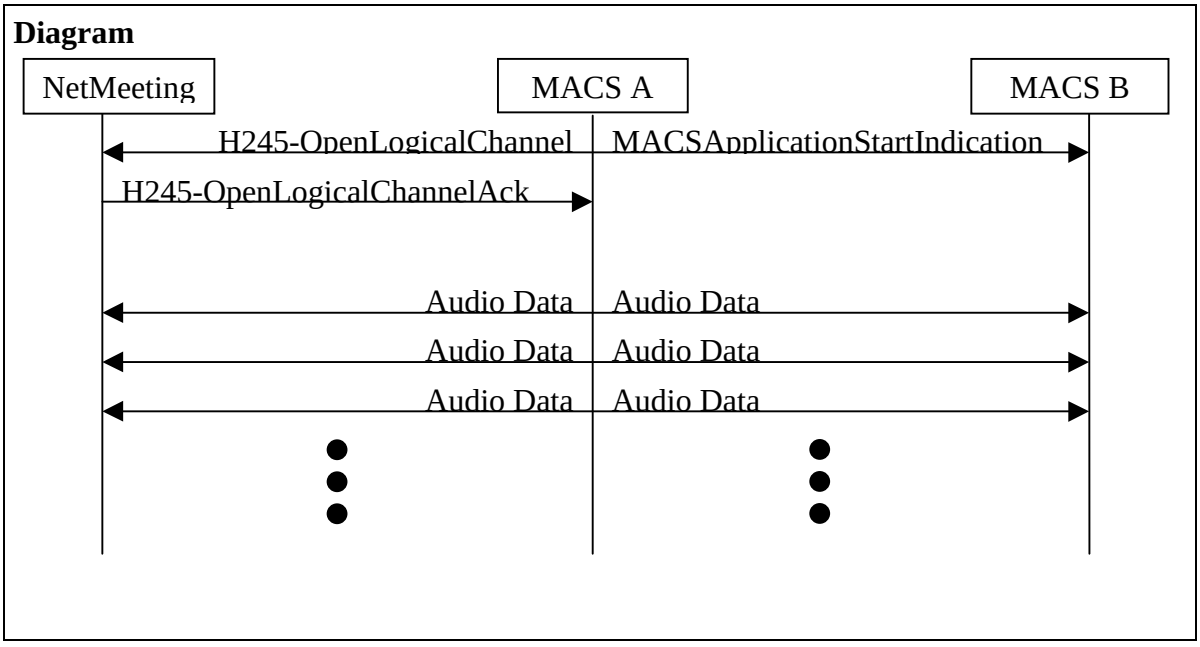


Figure 54: Message mapping for the application start (simple case)

This is the simplest case, as here the MACS instance with the enabled ITU interworking module (MACS A) starts the audio tool (here selected as an example, but the same sequence will apply to other tools) before NetMeeting. It is further assumed that this MACS (MACS A) is the current chair of the session, thus it will just announce the selected tool setup information to the others. For other MACS instances this is a simple indication containing the relevant data, while for the NetMeeting side a logical channel is established. Once this process is concluded the transmission of data by the tool can start, in this example multicast is assumed and thus message distribution is as shown directly to all parties of the multicast group.

Two further distinct cases exist. If the MACS instance running the ITU interworking module is not the session chair (case 1b and 2b) it will send an ApplicationStartRequest to the session chair, receive consequently an ApplicationStartResponse followed by an Application-StartIndication (which is equally true for case 3b). The latter is translated by the interworking module into a message exchange with NetMeeting for establishing a logical channel, leaving the following transmission of audio data as such unchanged. The second case (case 1b) arises if NetMeeting initiates the audio transmission by starting its audio tool, as shown in Figure 55. Here it is now assumed that the MACS instance with the interworking module (MACS A) is not the session chair, thus the above described additional exchange of messages regarding the application start becomes necessary. This is the most complex case for the application start which can arise. NetMeeting requests a logical channel as it wants to start the audio tool. This is translated into an ApplicationStartRequest sent to the session chair. The ApplicationStartResponse will be translated into the corresponding acknowledgment for the setup of a logical channel. If this fails at any stage the process is terminated. Assuming successful setup the session chair will now announce the new application within the session with an ApplicationStartIndication. This is translated by MACS A into a flow control command to NetMeeting. Only now is the audio tool of NetMeeting actually allowed to send data into the established logical channel. This additional message is necessary in order to allow the rest of the system to conclude its setup without already receiving data which can not yet be processed.

The same process as shown here for the example of the audio tool applies for any other tool used in the session. This concludes the most important processes of the session setup and the application start as needed in order to interact. Further messages exist and are handled by the system (see appendix A.4), as, for example, for stopping a tool or for leaving/terminating a session [78]. It is again important to note, that not all MACS messages can be mapped to corresponding ITU constructs, as not all of MACS features are supported.

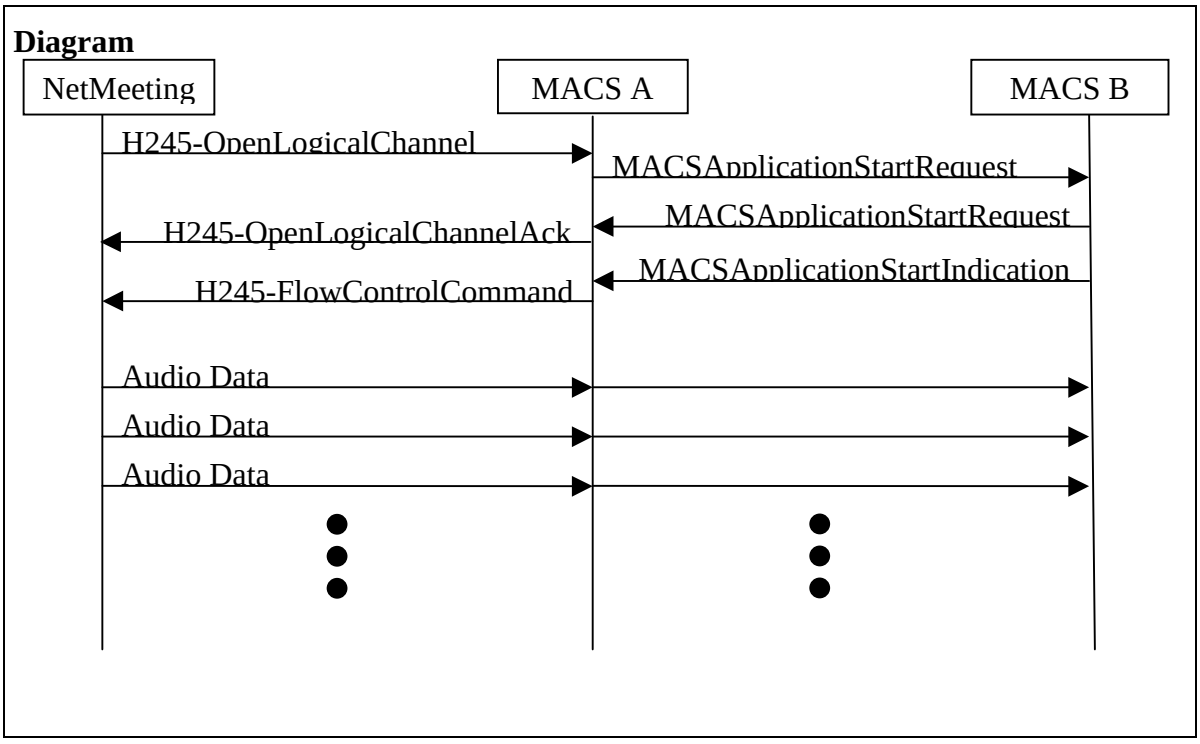


Figure 55: Message mapping for the application start (complex case)

Communication overhead

As mentioned earlier the ITU-T.120 is based on a tree structure of point to point connections, where each individual update is passed up in the tree, potentially merged with other updates before being integrated into the top nodes database. Finally a update message is send by the top node to all other nodes, informing them about the changes in the database. This is not an efficient way to distribute data. MACS on the other hand assumes normally a multicast network, making the distribution of data much more efficient.

The session is managed by a central instance, both on the MCU and GCC level, with the nodes organized in a tree structure. Complete copies of the configuration are kept in all nodes. A change is propagated upwards, possibly merged with other changes. The top level will then send an update down the tree.

5.3.6 Evaluation of ITU interworking design

The ITU interworking module shows the flexibility of the module approach taken in the MACS framework. Still a number of important limitations exist in respect to interworking with ITU based systems, specifically as here shown, with NetMeeting. These are mainly based on the fundamental differences of the approaches taken by MACS and the ITU standards. For the available media only audio and video are treated in this example. Chat and whiteboard are incompatible and, thus, would need a media convert for interworking. The application sharing of NetMeeting is in its concept not compatible with the idea of a MACS applications. While larger meetings with NetMeeting are theoretical possible it is unlikely that these are common or even really practical. As a result it is assumed that mainly single NetMeeting users will join a MACS session. Otherwise the load on the system running the MACS instance would increase considerably, especially if the audio and video streams have to be rerouted via the RTP-forwarder from an incoming unicast connection to a multicast group and vice versa. It is not envisaged that a MACS user will join a NetMeeting session,

though this is in principle possible. A NetMeeting user joined to a MACS session will be limited to audio and video, as the other tools are not compatible, as explained above. Further, it will not be possible to use floor control for the NetMeeting user. The MACS user will only receive limited information about the NetMeeting user, as the H.323 standard does not contain the same extensive information about a user as MACS provides. To provide a more complete integration and thus extended interworking capabilities between MACS and ITU based systems it would be advisable to create a separate entity working as a gateway and media converter. This entity could then be placed independently of any users MACS instance on a powerful server and, thus, could cater for significantly more than just a single person. To achieve this the current interworking module would have to be extended to form the core of a separate server application. This should be relatively easy to achieve. The implementation of the media converts on the other hand is a very complex and rather time consuming task, due to the sheer size of the involved standard families as covered in the ITU-H.323 [54] and T.120 [55] family of standards.

5.4 Interworking with typical MBone tools

The MBone is, as explained in section 5.1.2, a virtual overlay network over the Internet. While as such it is not a standard itself, a number of CSCW relevant standards have been developed and are typically used by MBone applications. The SAP, SDP standards allow session discovery. The MBone is in contrast to NetMeeting oriented towards a session centric view and not a user centric one. Session discovery is done via the session announcements send in the session description format (offline or indirect discovery of the sessions, e.g., via email or news are as well possible). A further protocol, SIP, allows to invite a user into an existing session. This is mainly aimed at providing the contact information. SIP partly handles user discovery, as it tries to resolve, for example, email addresses to real user login information. The applications in a typical MBone setup run independently of each other resulting normally in a lack of coordination, synchronization and control functionality (access control, floor control, etc.). Some developments in this area are aimed at changing this (e.g., MBUS see section 2.5.3.2).

The following sections will discuss the requirements for interworking between MACS and MBone, as well as the design goals and the structure of the MBone interworking module for MACS. Finally an evaluation of the selected approach is given.

5.4.1 Requirements for MBone interworking

Again the most basic requirements are the ability to gather session setup information and to execute the session setup itself. As there are many ways session setup information can be distributed it is difficult to consider all of these ways. The most common approach though is through session announcements [44] [45] send via a dedicated multicast group. This group should therefore be monitored to extract information about other sessions and should as well be used to announce the locally created sessions. The session announcements will contain the media used in the session, as well as the protocols and addresses used by the corresponding tools. This will allow the local system to match a local tool to each media and start it with the corresponding address and setup parameters. Thus, a local configuration must exist to do this mapping of media types to tools. Some examples of such media descriptions as used by SDP [45] are given in Table 20.

m=	**<media>**	**<port>**	**<transport>**	**<format list>**
m=	audio	49170	RTP/AVP	0
m=	video	51372	RTP/AVP	31
m=	application	32416	udp	wb

Table 20: Media types as defined by SDP

Multiple entries for each media may exist. **Transport** defines the protocol used, as, in these examples, RTP using the Audio Video Profile (AVP) or UDP. The actual coding format is then given in the **format list** in form of a number (e.g., audio format 0 is 8kHz mono).

5.4.2 MBone interworking design goals

The design goals are the same as for the ITU interworking module and, thus, the same comments as before apply. Here only the question of tool integration is somewhat different. There are no MBone specific tools as such. The tools such as audio and video use standard protocols and can directly interact with corresponding MACS tools regarding their basic features once address and coding information have been obtained. MACS tool specific features, such as the floor control as realized via the MACS control, will fail, that is, these functions will not be available. Tools using non-standard protocols will, in general, not allow for interworking with MACS. An example is the whiteboard **wb** often used on the MBone. An alternative here is to use the MACS whiteboard together with the MBone tools.

5.4.3 Analyzing the differences between MACS and MBone

Neither the MBone nor the typical MBone tools such as sdr, vic, vat and wb are specifically designed for CSCW. They can be used for such applications but only provide the most basic support. Especially the missing integration of these tools concerning, for example, synchronization and access control is of great relevance to the CSCW sector. As a result no real session control protocol commonly exists, though some approaches are under investigation [12] [79].

For the interworking of MACS with the MBone tools the opposite approach as previously used for the ITU interworking module is applied. Not MBone session participants are joined to a MACS session, but MACS allows to join MBone sessions. Due to this approach and, as previously stated, the absence of a control protocol the main problem to solve in order to enable interworking is the information extraction of session (and application) setup information.

The main two approaches to obtain this information are to listen to the session announcements packed with SAP [44] and SDP [45]. This approach is implemented in MACS. The other approach is to process SIP [46] messages (for a comparison of SIP vs. H.323 see [86]). While this approach is envisaged for MACS it is currently not implemented in MACS.

Thus, having a look at the different mechanisms to distribute this setup information for the MBone and for MACS, as well as the differences in content, the following points arise:

	MBone	MACS
distribution of session setup information	session announcements via SAP/SDP or SIP	session announcements via proprietary protocol or interworking modules
session announcement types	• new session • delete session • update session info • alive	• new session • delete session • update session info • alive

announcement contents	• for each application
	  • media description
	  • addresses
	  • coding
		• for each application
		  • media description
		  • addresses
		  • coding
		• session control info

Table 21: Differences MBone to MACS for session information

As Table 21 shows there are many similarities between MACS and the MBone mechanisms and tools (for details on the protocols used in MACS see section 3.4.5.2). The main difference is based on the integrated approach of MACS which allows a better coordination of the applications themselves and the session control, as not available with the normal MBone tools.

5.4.4 Structure of MBone interworking module

The structure of the MBone interworking module and its integration in MACS are similar to the ITU interworking module. The main difference is that due to the lack of session control in MBone, a session control of MBone users is not possible and therefore the setup results in a session without floor control. As in the typical MBone session there is no control instance the applications get directly launched given the parameters extracted from the session announcement. The applications will then interwork directly (if possible) without the involvement of a control. As a result the MBone interworking module only needs to monitor the session announcements to determine the availability or changes in available media within the session. This results in the relatively simple structure shown in Figure 56.

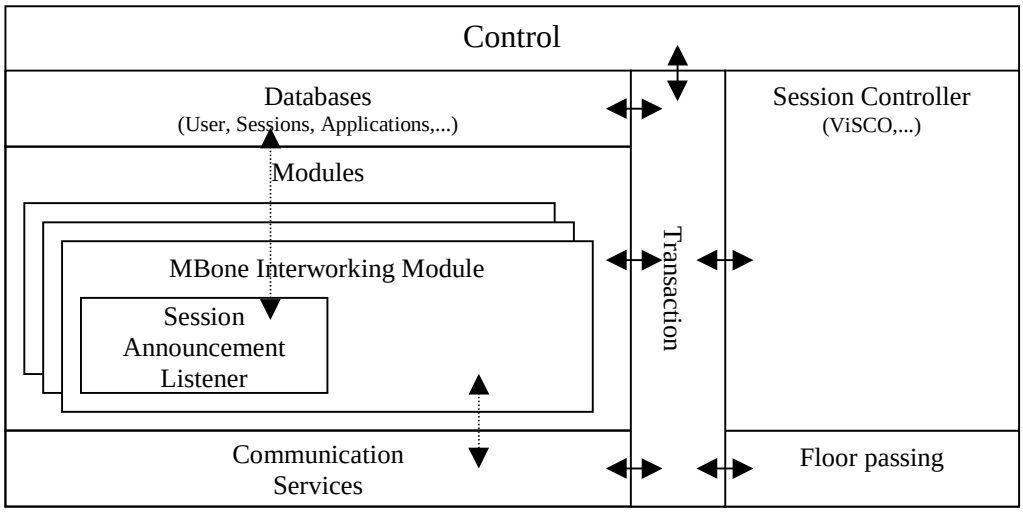


Figure 56: MBone interworking module

5.4.5 Evaluation of MBone interworking approach

The module design is straight forward and uses the standard integration into MACS. Again a number of limitations do exist due to the difference in nature between MACS and the MBone tools. As no single MBone tool exists it will depend on the users tool how far he can interwork with a MACS system. Typically he will be using a tool to choose a session out of the list of announcements, and then start audio, video and optionally a whiteboard. This is currently possible, only the whiteboard support is limited to the MACS whiteboard. The MACS whiteboard can be used standalone with MBone tools such as *sdr*, *vic* and *vat* (this setup was used repeatedly for a lecture). For a MACS user the same limitations will apply when joining a MBone session, additionally the advanced features of MACS will be disabled. To use other tools a media convert would be necessary, as explained before for the ITU interworking module. Basically interworking with the MBone tools is straight forward, needing little support and mainly results in dropping the advanced features of MACS.

6 Selected implementation aspects

This chapter discusses the main implementation aspects of the MACS framework. Hereby first general issues are considered, such as the implications of using Java as an implementation language. Further the three main parts of the framework, the application, module and net sections are discussed[4], as these provide most of the frameworks flexibility. Here the aim is to show how their integration into the MACS framework is done by giving a guide for integrating ones own modules. This is in each case illustrated by an example. As the point is to show the integration aspects the examples chosen are normally not very complex ones, except for interworking one, as here some complexity for illustrating the available possibilities is necessary. Then this section takes a look at the performance of the transaction system. The transaction system is the heart of the control in MACS and therefore will have a strong effect on overall performance and hence scalability. Finally experiences from a real world usage scenario are presented.

6.1 General implementation issues

For the MACS framework and all its components an important issue is portability. This did lead to the choice of Java as the implementation language. It is possible to integrate non Java components via JNI (Java Native Interface), but this approach is discouraged, as it is not be portable. The target platform for MACS is the JDK (Java Development Kit) V1.2. A number of further packages are used by the different modules as listed in Table 22. The main aim hereby was to use existing technology (actually in most cases technology under development and targeted to become a standard in its sector) in order to achieve the best possible deployment situation.

package	short description	relevant component	status
LRMP	a light weight reliable multicast protocol	Network Service (one of the offered underlying protocols)	optional
JMF	the Java Media Framework for audio and video support	Media tools (audio and video sender and receiver)	required for audio and video
JNDI[5]	provides access protocols to naming and directory services	ILS (Internet Location Server) module (part of ITU interworking set of modules)	optional

Table 22: Java packages overview for MACS

The choice of Java as an implementation language results in a number of problems. A important point is resource consumption especially in form of required memory, as well as the relatively slow execution speed.

[4] The code base as of June 2000 is assumed for the discussion. As MACS is an ongoing research project some details are likely to change.

[5] JNDI will starting with JDK V1.3 no longer be a separate package (release due in Q3/00).

Speed improvements are possible by using technologies as JIT or HotSpot and by efficient coding. JIT interpreters for Java only translate the code to be executed once. This is done right before the code is executed the first time, then the code is stored in the translated format and thus greatly increasing execution speed the next time the same code is accessed. HotSpot has a somewhat different approach. Here a profiler analyzes the code being executed. Code which is executed often will receive more and more optimizer time, not just storing the interpreted result, but as well trying to optimize the code itself.

Concerning the memory requirements it is important to note the JVM itself requires a great amount of memory before even starting the execution of the custom program. This is one of the reasons all of MACS components are run in the same Java Virtual Machine (JVM). This helps to improve performance and reduce resource usage. Table 23 shows a comparison[6] of this approach, that is to run all modules in the same JVM, in comparison to run each application module in a separate JVM.

memory usage	**single JVM**	**a JVM each**	**difference**
MACS	15.336 KB	15.336 KB	0 KB
MACS+ whiteboard	23.448 KB	36.508 KB	21.172 KB
MACS+ whiteboard+ TelSEE (see 2.1.4.2)	30.208 KB	61.648 KB	25.140 KB

Table 23: Memory usage comparison of single vs. multpile JVM approach

The overhead of the JVM for separate execution of application modules is quite clearly visible. A further advantage of running all modules in the same JVM, (each as its own thread) is that the communication overhead is much smaller. Direct calls are possible and no IPC (Inter Process Communication) is required.

6.2 Application integration

This section has a look at one of the key aspects of the MACS framework, the integration of application modules. Especially flexibility and control issues are here of importance. First this is treated in a general application module independent manner, followed by an example. The example gives a detailed description of the integration of the chat application module into the MACS framework. The chat module was chosen, as it is from the application specific aspect rather trivial and, thus, does not add unnecessary complexity to the integration issues.

6.2.1 General aspects of application integration

Flexibility is a key issue for the integration of application modules, thus first the configuration for the application module integration is explained. Then the integration into the control and the access to control services via its APIs is shown.

[6] These measurements were done on a Linux 1.2.30 system using JDK 1.1.3 and Swing 0.6.1 running on a Pentium 200MHz with 64MB RAM.

6.2.1.1 Configuration

Application modules are defined by implementing the *MacsApplication* Interface. This interface allows the control to handle these modules. They are entered into the configuration file to make them known to the control, as seen in Listing 1. Here the Chat, Whiteboard, AudioSender, AudioReceiver and Feedback application modules are included. For each of these at least a class implementing it has to be provided. They may further use their own set of entries, but these are not defined by the MACS framework.

```
# application module list
control.applications=Chat Whiteboard AudioSender AudioReceiver Feedback
# class and name definitions
Chat=de.tubs.macs.applications.chat.Chat
Whiteboard=de.tubs.macs.applications.whiteboard.Whiteboard
AudioSender=de.tubs.macs.applications.media.AudioSender
AudioReceiver=de.tubs.macs.applications.media.AudioReceiver
Feedback=de.tubs.macs.applications.feedback.Feedback
```

Listing 1: Control configuration for application modules

This information is used to load the corresponding classes and to provide GUI components labeled with the corresponding names for starting the specific application modules. The dynamic class loading mechanisms of Java are used in this process. Once the class is loaded it is checked for compatibility. This means it has to implement at least the *MacsApplication* interface and it has to provide a matching constructor. Further, version information is extracted and can be used to check for additional dependencies or compatibility issues. The class is stored in the application database and then instantiated in each session as required.

6.2.1.2 Control integration and service APIs

In a running session the start of an application will result in the invocation of an instance of the class implementing that application, as defined in the configuration. This instance is entered into the session database as part of the ongoing session.

Each application can access the services of the control. In order to minimize required variables these are implemented as public static variables offered by *MacsControl* and can thus easily and efficiently be accessed by all components of the system. The main services and some variables of general interest are given in Listing 2.

```
public static User localuser;                    // access to data of the local user
public static Log log;                           // unified logging facility
public static Resource macsResource;             // access to configuration/preferences
public static SessionManager sessions;           // access to session manager (i.e., session database)
public static UserManager users;                 // access to user manager (i.e., user database)
public static ApplicationManager applications;   // access to application manager (i.e., appl. database)
public static ModuleManager modules;             // access to module manager
```

Listing 2: Access to control services

The *localuser* is a reference to the corresponding user entry in the user database making its access more convenient.

The *log* (and thus debug) facilities allow the system wide logging. This enables to pipe log messages for all modules to a file, suppress them or apply a global configuration to them. This is used especially during debugging to allow component specific messages, though for the expert user some of these messages might as well be of interest. The debugging output can separately be enabled for many different components, according to the setup stored in the user configuration file. This allows, for example, each developer to only enable the debugging messages of interested for him. For the non-expert user this aspect of the system is hidden, as the default is to disable all debugging messages.

The access to the system and user configuration, as stored in the configuration files is provided via *macsResources*. The system does load the system configuration and user overwrites during initialization requiring no special treatment for the application modules to extract their own configurations.

The main interest for the application modules though will be the database access. This is provided via the *SessionManager* for session information, the *UserManager* for user information and *ApplicationManager* for application module information. These managers provided through the main control handle the system wide databases. Some information is only session relevant and thus included in the session entry (see structure of database in section 3.4.4.1). These are accessed via a similar mechanism, that is a manager provided as a public variable within the session frame. Thus, in the *Session* class the following access points as shown in Listing 3 are provided.

```
public SessionNet snet;                          // access to Communication Services
private FloorControl floor;                      // access to floor control (has access functions in session)
public SessionApplicationManager applications;   // manage applications in the session
public List users;                               // manage users in the session
```

Listing 3: Access to services in a session

Thus for the database access the three global classes *SessionManager*, *UserManager* and *ApplicationManger*, as well as the session specific one, *SessionApplicationManager* are relevant.

Having a look at the API of the *SessionManager*, as shown in Listing 4, the basic functionality of adding a new session and removing existing sessions is found, as well as a call to execute an expire on all sessions. This is done periodically in order to expire sessions for which no update message (see section 3.4.5.2) has been received during a period of time exceeding the configured timeout. If a session is expired it is assumed that the session did end (possible due to a host or network failure).

Further some methods are defined to get a list of sessions, retrieve session netries by their ID or name and to identify their type.

```
void addSession(Session session);       // add a session to the database
void expireSessions();                  // check all sessions for an expire
Vector getAllSessions();                // get list of all sessions
Vector getLocallyCreatedSessions();     // get list of all locally created sessions
Session getMacsSession(ObjectID id);    // get a session by its ID
Session getMacsSession(String name);    // get a session by its name
boolean isMacsSession(String name);     // determine if the given name belongs to a MACS session
void removeSession(Session session);    // remove a session
```

Listing 4: Database session access (SessionManager) API

For the access to the user database, as seen in Listing 5, a similar set of methods is available. Again it is possible to add new users, to remove old users or to run an expire. Further a list of known users can be retrieved or just a single user as identified by his user ID.

```
void addUser(User user);                // add a user to the database
void expireUsers();                     // check all users for an expire
User getUser(ObjectID id);              // get a user by its ID
Vector getUsers();                      // get list of all users
void removeUser(User user);             // delete a user from the database
```

Listing 5: Database user access (UserManager) API

For the access to the application database independently of any session only very few methods are provided, as here, for example, no dynamic addition of applications is currently supported. These functions can be seen in Listing 6. It is at this level only possible to get a list of existing applications, to retrieve application names by ID and (for the control only) to invoke a new application instance, which will then be bound to a session (not shown here). All further functionality is provided in the frame of the session.

```
String getApplicationName(ApplicationID id);     // get name of application by its ID
ApplicationID[] getApplications();               // get list of applications
protected MacsApplication newApplicationInstance(ApplicationID id);     // get application instance
```

Listing 6: Database application access (ApplicationManager) API

Listing 7 shows the methods as provided in the session via the *SessionApplicationManager* to handle applications. These allow to test if an application exists (i.e., if it is locally installed, as there is no guarantee that all participants have the same system setup) and if it is in use. It is as well possible to get a reference for a given application, which is necessary if a direct interaction is intended. Further, it is possible to obtain the ID of the application and to get the network addresses used by it. As an application can use multiple addresses these are identified by a label (e.g., for the audio sender possibly a "left" and a "right" channel, each using a different address).

```
MacsApplication appGetReference(ApplicationID id);                // get application by its ID
boolean applicationExists(ApplicationID id);                      // determine if application exists
NetAddress applicationGetAddress(ApplicationID id, String label); // get specific address
ApplicationID applicationGetID(MacsApplication application);      // get the ID
boolean applicationUsed(ApplicationID id);                        // determine if instance exists
```

Listing 7: Database application access in session (SessionApplicationManager) API

The main interest of the application module is normally in the provided functionality within its current session. Listing 8 shows an extract with the key functionality provided by the session. These can be further subdivided into three basic types of functionality:

- session description
- session mechanisms
- handlers

The session description methods allow to set and retrieve data specifying the session, as, for example, its creator (*getCreator()*), type (*getType()*), name (*getName()*) and the like. These are not all included in the extract. For a more detailed overview the complete MACS API should be consulted [67].

Next the session mechanisms are accessible. These are normally used by the controller to handle invite, join, leave or similar actions, but these might as well be used by an application module. A good example is provided by the introductory scenario assuming the use of a CAD application in the automobile sector. If here a conflict in a drawing is detected the CAD application module would then automatically try to invite all users who are involved in this conflict into a session in order to resolve the detected conflict. Depending on the session parameters this can happen automatically or the application will initiate the invite process and the session controller will take over (and might get a local confirmation form the user before actually inviting the other users).

The third set of methods are handlers. These are used to pass messages between instances of MACS but are not really relevant to the control itself. These cover the application, floor control and the direct session controller communication.

One special case remains to be noted. A session object exists already once a session announcement has been received. Such an object only contains the session description and has no further functionality by itself. Only once the session is joined by the local user the session object will provide the complete session functionality. To locally created a session and, thus, to initiate the sending of announcements for this session the *createLocalSession()* method is provided.

```
Status acceptInvite(...);                          // accept an invite request
boolean applicationStart(ApplicationID id);        // start an application
Status createLocalSession(...);                    // create a new session
boolean expelUser(ObjectID user);                  // expel a user
User getCreator();                                 // get the user who created the session
ObjectID getID();                                  // get the ID of this session
String getName();                                  // get session name
String getType();                                  // get type of session
Vector getUsers();                                 // get list of users
boolean hasUser(ObjectID userID);                  // check for a specific user in session
boolean inviteUser(User current);                  // invite a user
boolean inviteUsers(java.lang.Object[] users);     // invite users
boolean isLocalSession();                          // check if the session was created locally
Status joinRemoteSession(...);                     // join a session
boolean leave(User user);                          // leave the session
void modifyUser(ObjectID userID, User user);       // change user info
void quit(MacsApplication application);            // quit an application in the session
void receiveApplicationData(Object object);        // receive data for the applications
void receiveControllerData(Serializable object);  // receive data for the controller
void receiveFloorData(Object object);              // receive data for the floor control
boolean sendData(Serializable data);               // send controller data
```

Listing 8: Extract of functionality provided via the Session

This concludes the main points of how an application module can access the control services and the session functionality, except for the floor control, which is described in [19].

Taking one step back and having a look at how the control accesses the application module we must consider the *MacsApplication* interface, as shown in Listing 9. Each application module must implement this interface and the main class of the application module as defined via the configuration will be checked for compliance with this interface. The interface allows basic access to the application module, providing name and version information, as well as initialization, start and exit methods. Further handlers to pass session and floor events are provided.

```
void exitApplication();              // called to exit the application
int getVersionMajor();               // get version info
int getVersionMinor();               // get version info
String getVersionName();             // get name of application
boolean initApplication(...);        // initialize application module
boolean processFloorEvent(...);      // deal with floor indications
void processSessionEvent(...);       // deal with session events
boolean startApplication();          // actually start the application module
```

Listing 9: MacsApplication interface

6.2.2 The chat example

Having considered both the available functionality provided by the control and the session to an application module and the functionality the application module does provide itself, this section will use the chat application module to actually show a practical example of the integration of an application module into the MACS framework. This can as well be used as a form of template for integrating ones own application modules into MACS.

The first step is to create the main class of the application module implementing the *MacsApplication* interface. Normally, though not necessarily this class will be included in the MACS directory tree as an application and, thus, form a separate package in the application section. For the chat this is the package de.tubs.macs.applications.chat, as shown in Listing 10. Further does the Chat module implement the *NetReceiver* interface for using the network services to actually transmit its data (i.e., text typed). The *MouseListener* and *KeyListener* interfaces are implemented in order to handle floor request via the provided button and user typed text (see Figure 34 for a screenshot including the chat application module).

```
// Chat.java
//--------------------------------------------------------------------------
package de.tubs.macs.applications.chat;
//--------------------------------------------------------------------------
/**
 * A Chat application (as an simple example for a MACS application).
 **/
public class Chat implements MacsApplication, NetReceiver, MouseListener, KeyListener
{
}
```

Listing 10: Chat application module declaration

Starting with the implementation of the *MacsInterface* the version information has to be provided, as shown in the source code extract in Listing 11.

```
// const
  public static final int VERSION_MAJOR=0;
  public static final int VERSION_MINOR=5;
  public static final String VERSION_STRING="Chat";
/**
   * method to return the major version of the application, e.g.
   * X for Version X.Y
   */
  public int getVersionMajor()
  {
    return VERSION_MAJOR;
  }
  //-----------------------------------
```

```
 /**
  * method to return the minor version of the application, e.g.
  * Y for Version X.Y
  */
 public int getVersionMinor()
 {
   return VERSION_MINOR;
 }
 //---------------------------------
 /**
  * method to return the version name of the application, e.g.
  * for "Chat V0.1" is "Chat"
  */
 public String getVersionName()
 {
   return VERSION_STRING;
 }
```

Listing 11: Chat MacsApplication interface version implementation

Next the initialization, start and exit methods for the *MacsAplication* interface have to be implemented. The initialization will occur once the class has been loaded and been associated with a session. The start will only occur later, once the control has concluded the setup and finished the integration of the chat application into the session. In the case of the chat application module this is rather simple, as chat does no internal bookkeeping and, thus, does not need to start its own thread. The complete setup is therefore done during the initialization. This includes the creation and arrangement of the GUI components as shown by the extract contained in Listing 12.

```
/**
   * The Chat init. This initializes the application and makes it ready to run (start).
   *
   * @param session the session this application runs in
   * @param control the control this application runs in
   * @param display the display component used by this application
   */
  public boolean initApplication(ApplicationID id, Session session, JComponent display)
  {
    int lines;
    int width;
    NetAddress group;
    // read configuration
    lines=MacsControl.macsResource.getInt(VERSION_STRING+".lines");
    width=MacsControl.macsResource.getInt(VERSION_STRING+".width");
    // setup GUI
    display.setLayout(new BorderLayout());
    list=new JTextArea();
    list.setRows(lines);
    list.setColumns(width);
```

```
  [...]
  // network
  NetAddress address=session.getApplicationAddress(id,"main");
  try
   {
     net=MacsControl.netManager.getNet(this, address);
   }
  catch(NetException ex)
   {
     MacsControl.log.error(VERSION_STRING+".<init>","error creating net: "+ ex);
   }
  // floor handling
  floor = new Floor(session, this, "Chat", Floor.CATEGORY_FREE);
  floor.setActive();
  // done
  return true;
}
//-----------------------------------
/**
 * method called by MacsControl to shutdown this application
 */
public void exitApplication()
{
  floor.exit();
  net.close();
}
//-----------------------------------
/**
 * method to start the application
 */
public boolean startApplication()
{
  return true;
}
```

Listing 12: Chat MacsApplication interface main implementation extract

As can be seen clearly the main code is the initialization. Here basically four operations occur:

- reading of configuration information (lines and rows to be used by the text area)
- the GUI setup
- the setup for access to the network services
- the setup for access to the floor control

The first two points are sufficiently explained by the code in Listing 12. The setup for access to the network services is a good example on how to handle these. The network service implementation and usage is explained in detail in section 6.4. The setup for access to the floor control includes three steps:

1. A floor object has to be created. In this case it is associated to the right to send chat text.
2. The floor object has to be activated.
3. Finally, the received floor messages have to be handled by the *processFloorEvent()* method of the *MacsApplication* interface

This is shown in Listing 13. Here only the gaining and losing of the chat floor is treated, as the chat application is not itself able to handle holder or policy questions. These are normally handled by the session controller [19].

```
// floor method of MacsApplication interface
  public boolean processFloorEvent(int type, long id)
  {
    if(id==floor.getId())
      {
        switch (type)
          {
          case Floor.FLOOR_GAINED:      have_floor = true; break;
          case Floor.FLOOR_LOST:        have_floor = false; break;
          case Floor.HOLDER_CHANGED: break;
          case Floor.POLICY_CHANGED:  break;
          }
        setFloorLight(have_floor);
      }
    return true;
  }
```

Listing 13: Chat MacsApplication interface floor implementation

The chat application module is not interested in session events. If a more IRC (Internet Relay Chat) conform implementation would be wanted these could be used to detect other participants starting their chat application modules and, thus, the typical "User X has entered the chat channel" for a new user or corresponding leave messages could be displayed.

The actual chat code is rather simple it is just a question of taking the user input and sending it through the network services, as setup during initialization, and displaying received chat text. This is shown in Listing 14, as well as the handling of the chat floor.

```
 /**
  * Get chat text (KeyInterface).
  */
 public void keyTyped(KeyEvent e)
 {
   if((e.getKeyChar()=='\r')||(e.getKeyChar()==KeyEvent.VK_ENTER))
     {
       if (have_floor)
         {
           floor.resetTimeout();
           send(talk.getText());
           linesprinted += 1;
           talk.setText("");
         }
     }
 }
 /**
  * Receive incoming text (NetReceiver Interface).
 **/
 public void receiveObject(Object object, int size, Net net)
 {
   receiver((String)object);
 }
 // send entered text
 private void send(String message)
 {
   String str=MacsControl.localuser.name+": "+message;
   try
     {
       net.send(str);
     }
   catch(NetException ex)
     {
       MacsControl.log.error(VERSION_STRING+".send","error sending: "+ex);
     }
   receiver(str);
 }
 // internal handling of received (and send) text
private synchronized void receiver(String message)
 {
   if(list.getText().length() > 0)
     list.append("\n" + message);
   else
     list.append(message);
   list.setCaretPosition(list.getText().length());
 }
```

Listing 14: Chat main code

Having covered the main points of the code for the chat application module only the actual configuration remains to be examined in order to successfully conclude the integration of chat into the MACS framework. As explained in section 6.2.1.1 the name of an application module must be entered into the list of available modules, its main class implementing the *MacsApplication* interface has to be specified and optional parameters can be provided. Listing 15 shows the chat entries in the default MACS configuration files.

```
# application-list
control.applications=Chat Whiteboard AudioSender AudioReceiver Feedback
# applications
Chat=de.tubs.macs.applications.chat.Chat
# the chat application
Chat.lines=10
Chat.width=40
Chat.greenLight.image=images/green_light.gif
Chat.redLight.image=images/red_light.gif
```

Listing 15: Chat application module configuration

6.3 Module integration

This section discusses the integration of modules into the MACS framework, starting again with the general aspects, followed by an example. Here the ITU interworking module was selected despite its complexity, as it uses a wide range of hooks into the system and therefore can be used to show a number of important aspects.

6.3.1 General aspects

The MACS framework offers an open interface to integrate modules into its control component. These can have a great variety of functions, as they can directly access the internals of the control. Despite that they are in their interfaces and the way they are handled similar to application modules their role is completely different. They are loaded during startup, if so specified, and run completely independent of any given session or application.

6.3.1.1 Configuration

The approach used for module integration allows a great deal of flexibility. This has to be supported by the configuration mechanisms. These are again very similar to the application module integration and are based on the same textual configuration files. Thus the addition or modification of modules is an easy task, as only the configuration files has to be modified. Listing 16 shows an example for a module configuration. Here the interworking modules are specified, that is the MBone interworking module and the ITU interworking module including ILS support. These are for the default system installation disabled and can be enabled in the users configuration file, which, as explained before, overwrites the system configuration.

```
# module-list
control.modules=mbone Itu Ils
# mbone-module
mbone.use=false
mbone.class=de.tubs.macs.control.modules.mbone.MBone
# ITU-gateway-module
Itu.use=false
Itu.class=de.tubs.macs.control.modules.itu.Gateway
Itu.callport=1720
Itu.endpoint.user.image=lib/de/tubs/macs/util/misc/images/H323User.gif
# Ils-module
Ils.use=false
Ils.class=de.tubs.macs.control.modules.ils.IlsService
Ils.providerURL=ldap://ils2.microsoft.com:389
Ils.ldapVersion=3
Ils.baseDN=o=Intranet
Ils.filter=(&(c=DE)(objectclass=RTPerson))
Ils.updateInterval=120000
Ils.category=1
```

Listing 16: Module configuration entries

Taking the example of the ILS module the corresponding selection in the configuration tab is shown in Figure 57. For each of the modules at least the "use" and "class" fields must be specified in order to enable the module and load the corresponding implementation.

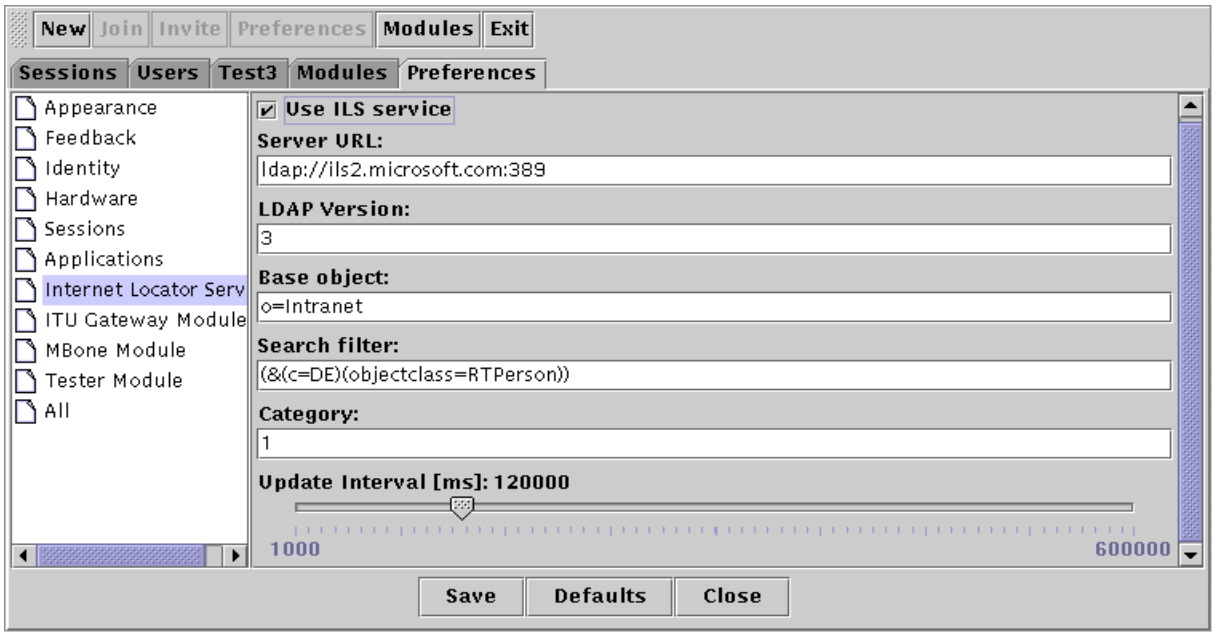


Figure 57: ILS module configuration

6.3.1.2 Control integration and further service APIs

If a module is marked for use it is loaded during system initialization. The sequence hereby is that the resource and database management will already be up and running before the modules

are loaded and initialized. The GUI on the other hand will only be partially initialized, such that the modules can access their own tab despite that it is not yet shown.

The class implementing the module is checked during its loading to confirm the implementation of the *MacsModule* interface, similar to the process for the application modules and the *MacsApplication* interface. The *MacsModule* interface, as shown in Listing 17, provides the control with basic access to the module. Again version information is provided, as well as mechanisms to start, stop (exit) and check the status of the module. Additionally a method is provided to get the panel used by the module to display its GUI elements. This method should only be called by the control during initial GUI building in order to place the module GUI into the corresponding module tab according to the selection in the module list (see Figure 32 and Figure 52).

```
int getMajorVersion();          // get version information
int getMinorVersion();          // get version information
String getModuleName();         // get module name
void exitModule();              //exit module execution
void startModule();             // start the module
String getModuleStatus();       // get status of module
JPanel getModulePanel();        // get the panel displayed in the module tab.selection
```

Listing 17: MacsModule interface

Modules are, in general, as mentioned before, deeper integrated into the system than applications. All the in section 6.2.1.2 described APIs are available. Especially for the interworking modules the communication services API is of interest, which was previously not explained, as it should not be accessed directly by an application module. The communication service API, as seen in Listing 18, provides status information, low level mechanisms and extended handler access.

```
void addSessionNetListener(SessionNetListener listener);          // add a new listener
int checkStatus();                                                // check status of session net
boolean expel(ObjectID user);                                     // expel a user
Status getApplicationAddress(ApplicationID app_id, String label); // get an application address
int getTTL();                                                     // get scope used by the session
String getType();                                                 // get type of net used by the session
Status invite(ObjectID user, long req_num, Serializable data);    // invite a user
boolean leave();                                                  // leave the session
boolean leave(User user);                                         // locally managed user leaves
void removeSessionNetListener(SessionNetListener oldlistener); // remove a session net listener
boolean sendData(Serializable object);                            // send controller data
boolean sendData(Serializable object, User user);                 // same for a locally managed user
boolean sendFloorMessage(object);                                 // send a floor message
boolean sendFloorMessage(Serializable object, User user);         // same for a locally managed user
boolean terminate();                                              // termiante session
```

Listing 18: Communication service (SessionNet) API

The status information includes the actual status of the *SessionNet*, as well as information about its properties, such as, for example, the scope of this session (i.e., TTL). The low level mechanisms are used by the control to manage a session or to exchange specific data between MACS instances within a session. Latter ones are a direct mappings from the in Listing 8 shown methods provided by the session. The others are the corresponding low level versions. For example, a user invite in the session will check with the session controller, update databases if needed, prepare parameters and only then call on the above listed invite method of the communication services. The handlers on the other hand are called directly, for example, by the interworking modules. These include adding and removing of listeners, as well as calls to low level mechanisms providing an explicit user reference (else the local user is assumed). This is necessary if an interworking module wants to emulate a user of a interworked system in a transparent way for the other MACS users, as explained further in the next section.

The control has a manager for handling modules, similar to the way applications, users and sessions are handled. The *ModuleManager* is mainly used during initialization, but might be needed during runtime to determine if a specific optional functionality realized via a module is available. A good example is the remote controllable camera. If such a camera is connected and the local user has enabled its support via the *CamControl* module, then the session controller can gain access to the services of this module via the *ModuleManager*.

```
void start();                          // start all configured modules
void exit();                           // exit all running modules
ModuleInfo[] getModules();             // get list of modules (info blocks)
MacsModule getModule(String name);     // get instance of specific module if available
```

Listing 19: Module access (ModuleManager) API

The control directly uses this functionality during initialization to start all configured modules and during shutdown to exit all running modules. The other functions are used to obtain a list of available modules (mainly used by the control) and to get an instance of a specific module if available. Thus, this allows access to the functionality offered by specific module, as explained for the example of the *CamControl* module (see [19] for more detail about this module).

6.3.2 The ITU interworking module

The design of the interworking with ITU based systems is described in section 5.3. Here the details of the implementation and integration of the ITU interworking module into the MACS framework are explained.

The ITU module handles the actual communication via the ITU-H.323 standard. It is responsible for establishing a connection, doing the session management during the conduction of the session and finally at the end of the session it is responsible for the session tear down. The basic establishing of the control channels is only one important point. Further address and compatibility information have to be managed in such a way, as to enable the MACS media tools to receive data from - and send data to - the NetMeeting user. To achieve this the gateway module allows a NetMeeting user to connect himself to a MACS instance, where he is displayed as shown in Figure 31. The user is entered into the MACS user list, replacing any previous entry (for example, created via the ILS service). Now the NetMeeting user can be invited into any MACS session just like any other MACS user. No differences are

noticeable in basic handling, but the reduced capabilities will become apparent within the session. A more detailed description of the sequence of events and some of the limitations is contained in section 5.3.4 and in [78].

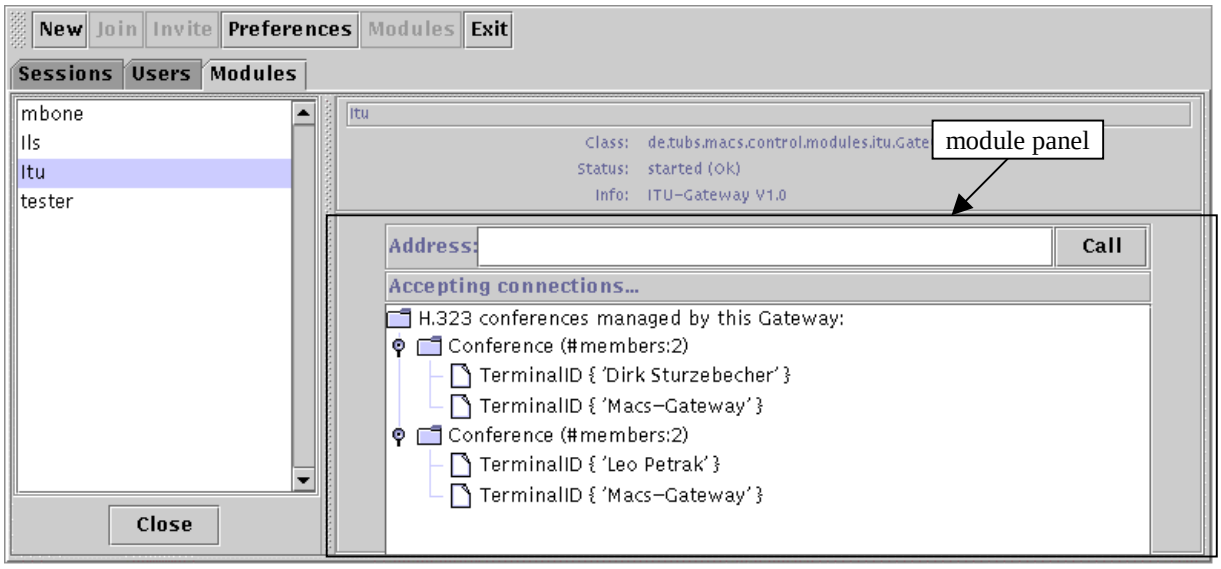


Figure 58: ITU-module integration into MACS

The first step in the implementation of a MACS module is to define its main class and to implement the *MacsModule* interface, as seen in Listing 17. The methods needed to conform to this interface are straight forward in their implementation, except maybe for *getModulePanel()*. This call expects to get the panel used for the GUI components of the module. Figure 58 shows this for the ITU interworking module. Listing 20 shows the definition of the ITU interworking module.

```
// the package
package de.tubs.macs.control.modules.itu;
// the main class
public class Gateway extends Thread implements MacsModule
{
  [...]
}
```

Listing 20: ITU interworking module definition

Next the module must be included in the configuration. Listing 16 shows such a configuration actually including the ITU module.

The system is now able to load and start the module, if it was enabled by the user via his configuration. If the module tab is opened and the corresponding module selected, the GUI components of it will be displayed. In this case the ITU module it will provide feedback information about the managed systems and allows to place calls manually in order to contact a not yet connected NetMeeting system, as shown in Figure 58.

The basic module implementation is straight forward and it is easy to use this basic template to integrate a new module into the MACS framework. In The case of the interworking modules the interesting part is their close integration into the functioning of the control. The job of the ITU interworking module is to establish, or allow others to establish, a connection from NetMeeting with MACS. Once this connection is established the NetMeeting user is made known to the MACS users, just as if he was a MACS user. Thus, some system (that is in this case the system with the ITU interworking module) must emulate the NetMeeting user. As a result this system has not only to react on session messages (as defined in 3.4.5.2) for its local user, but as well for all users connected via the interworking module. To achieve this it will link itself into the message processing of the communication services, as already mentioned in 6.3.1.2 via the communication service API (see Listing 18). Now it can intercept all messages and send the required responses without any of the MACS users, including the local MACS user noticing any difference to the normal handling by MACS. The messages generated in this way have to be clearly distinguishable from the normal messages generated by the local user and thus the communication service API provides an alternative form for most methods with an explicit hand over of the user for whom the message is generated. Further information about the actual code and details on the protocol conversion is contained in [78] and not of interest for the principal integration of a module into the MACS framework.

6.4 Integration of protocols into MACS network services

As introduced in section 3.2 the network services must provide a flexible frame for different underlying protocols, hiding their specific details from the application modules in order to maintain overall modularity and flexibility of the framework. This section discusses first the general aspects of the integration of a net module into the network services of MACS, followed by a specific example, the LRMP module.

6.4.1 General aspects

As shown in Figure 14 the general manager instance has to handle generic address and receiver/transmitter objects. Thus, in the MACS network services such an management instance, called *NetManager,* exists. It can generate, check and handle generic addresses in form of *NetAddresses* (an interface which is implemented by the actual net specific address class). The receivers and transmitters are formed through the *NetReceiver* interface and *Net* object (again implemented by net specific classes). This allows a complete encapsulation of the underlying protocol, as shown in Figure 59.

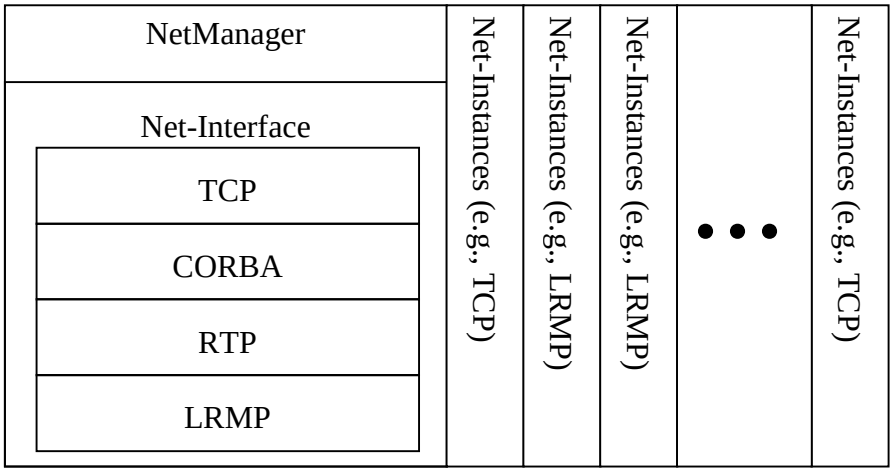


Figure 59: Network implementation structure

6.4.1.1 Configuration

The *NetManager* is configured via a text based property file. Listing 21 shows an example for a setup including TCP and LRMP support. As a minimum the name and the three implementing classes have to be defined for each type.

```
# available types
net.types=lrmp tcp udp
# lrmp
lrmp.name=LRMP Network
lrmp.group=225.000.000.000/7000
lrmp.options=15/100
lrmp.netclass=de.tubs.macs.util.net.LRMP
lrmp.addressclass=de.tubs.macs.util.net.LRMPAddress
lrmp.dialogclass=de.tubs.macs.util.net.LRMPDialog
# tcp
tcp.name=TCP Network
tcp.group=localhost/9999
tcp.options=centralized
tcp.netclass=de.tubs.macs.util.net.TCP
tcp.addressclass=de.tubs.macs.util.net.TCPAddress
tcp.dialogclass=de.tubs.macs.util.net.TCPDialog
```

Listing 21: Net configuration

This structure allows the easy addition of new implementations. These can then be used without any changes to the system, except adding the corresponding entries into the *NetManagers* properties file.

6.4.1.2 Class relations and supporting classes

Further some small support classes are needed. For example, it is desirable that an expert user can directly enter a session address, even if he did not receive an announcement (and thus can not click on it to join the session). This is not only relevant for joining sessions, but might as

well be necessary to setup non-standard parameter when creating a session. As each protocol has different parameters only the protocol implementation can provide a suitable dialog including its specific parameters. Each underlying protocol implementation in MACS, thus, has to provide not only a class implementing *NetAddress* (the generic address interface), *Net* (the generic interface for the main network class) and *NetReceiver* (the generic interface for receiving data send by others using the same net type and address), but as well a *NetDialog* (the above introduced option dialog) implementation. A number of such implementations currently exist, but basically only two are in use, as the others have been mainly created for testing purposes. These two are TCP and LRMP. A CORBA implementation did exist, but is no longer included in the active development tree. The RTP implementation is still under development, while a pure UDP implementation has a very limited usability, as segmentation of large data blocks is not supported and therefore large data block are simply truncated to the currently supported packet size. Each of these implements its own version of *Net*, *NetAddress* and *NetDialog*, as mentioned before.

The *NetManager* will ensure locally unique addresses. In order to ensure session or MACS wide unique addresses it is necessary to mark the addresses used by other instances. To achieve this the control has to extract all addresses from the received announcements and call on the *NetManager* to mark these as used.

When creating a new *NetAddress* the type/implementation to be used is determined via a string, which is matched to one of the entries in the *NetManager*s property file. Each *NetAddress* contains the type and the actual representation of an address of that type. Further an optional label is included. This allows the application to tag the *NetAddress* for a specific use. If, for example, an audio tool wants to transmit the left and right channel separately it can use two instances of *Net*. The receiving audio tool will on the other side be able to determine the two addresses used, but will need the tags to determine which address is used by which channel (left/right).

The *NetManager* will load the implementing classes at runtime. This prevents unnecessary memory allocation by not used classes and at the same time allows runtime modifications. It is possible that an application determines that a better suited (or newer) implementation of a network protocol is available, downloads it from a trusted site and installs it (all at runtime).

Figure 60 shows a class diagram for the network service level featuring the TCP and LRMP implementations. It shows clearly how the interfaces are used to hide the underlying details and how these are connected to the *NetManager*.

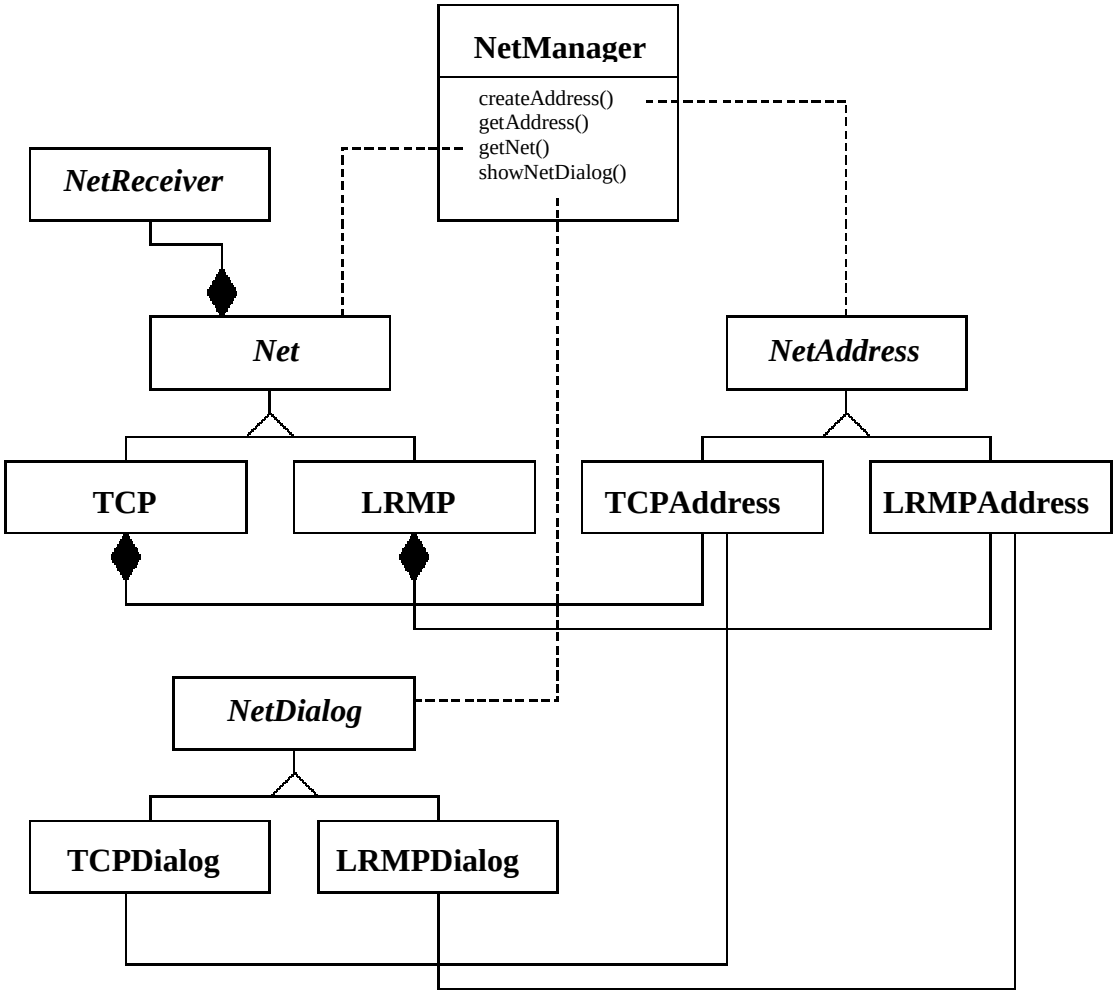


Figure 60: Network services UML class diagram

6.4.1.3 Control integration and further APIs

The *NetManager* provides within MACS the main API to the network services, as shown in Listing 22. It does provide access to setup the *NetManager* itself, and to determine its configuration by, for example, enumerating the existing (i.e., configured) network types. Further the address management is handled. This allows to create new, unused addresses for later usage, to mark addresses as known and to delete them again. Further, it is possible to check if a given address is used. Two further important methods exist, the one allows to actually create *Net* instance for a given address and the other can be used to obtain a user customized *NetAddress* through a dialog.

```
NetManager getNetManager();                              // get plain NetManager
NetManager getNetManager(Resource res);                  // get NetManager with specific setup
Enumeration getNetworks();                               // enuerate all known net types
NetAddress createAddress(String type);                   // create an address for a specifc type
NetAddress createAddress(String type, String label);     // as above, but using a specific label
NetAddress getAddress(String type, String group);        // retrieve a specific NetAddress object
boolean addAddress(NetAddress address);                  // add an address (e.g., mark it used)
void delAddress(NetAddress address);                     // delete an address
boolean addressUsed(NetAddress address);                 // check if an address has been used
boolean addressUsed(String type, String group);          // check if an address has been used
Net getNet(NetReceiver receiver, NetAddress group);      // create a new net instance
void showNetDialog(String type, Hashtable dialogs);      // allow custom address creation
```

Listing 22: NetManager API extract

Once a *Net* instance has been created via the *NetManager* it can be used to send data to other *Net* instances on the same *NetAddress*. The *Net* interface, as shown in Listing 23, further provides methods allowing to inquirer about its properties, such as its type, options or address used. To actually receive data from others on the same *NetAddress* a class implementing the *NetReceiver* interface, as shown in Listing 24, must be implemented. This will, in general, be the same class as the one requesting the *Net* instance, but sending and receiving capabilities might as well be separated.

```
void close();                // close the network connection
void flush();                // flusch internal buffers, if exist
NetAddress getAddress();     // get address of this connection
Object getOptions();         // get possible obtions of this connection
int getTTL();                // get the scope of this connection
String getType();            // get type of this connection
int send(Object object);     // actually send data to others
```

Listing 23: Net interface

```
void receiveException(NetException exception);          // handle exceptions
void receiveObject(Object object, int size, Net net);   // receive data send by others
```

Listing 24: NetReceiver interface

As can bee seen the *NetAddress* plays in all these interfaces a central role. The *NetAddress* interface, as shown in Listing 25, provides capabilities to compare two addresses. This is the key to allowing the marking and detecting of used addresses. Further it allows to retrieve the properties of the given *NetAddress*.

```
int compareTo(NetAddress address);    // compare two addresses
String getLabel();                    // get label of this address
Object getOptions();                  // get options associtated with this address
String getType();                     // get address type
void setOptions(String options);      // set options
```

Listing 25: NetAddress interface

6.4.1.4 General performance considerations

While the above described setup is a very flexible setup the question of performance has to be asked as well. This not just in respect to the described setup, but as well for the protocol and Java implementation overhead. A comparison of a ping command under Unix, with a direct Java implementation, a setup using the above described Net environment and the LRMP protocol, as well as TCP will give an estimate of the overhead. This comparison is shown in Table 24.

	delay	**comments**
Unix Ping	0,4ms	direct Unix command
Java Ping (UDP)	7ms	direct Java implementation
Java Ping (LRMP)	70ms	Java implementation using Net environment and LRMP
Java Ping (TCP)	80ms	Java implementation using Net environment and TCP

Table 24: Delay measurements of transport level implementations

The absolute value of the measurements are highly dependent on the network and system load, as well as on the versions of the tools used. The relative comparison on the other hand indicates that:

1. The Java ping is an order of magnitude slower than the Unix ping, probably due to the overheads of the JVM, such as object serialization and performed security checks .
2. The reliable transmission increases the ping overhead again drastically resulting in yet another order of magnitude increase in the measurements.

6.4.2 The LRMP example

This section takes LRMP as an example on how to integrate a protocol into the MACS network services. Thus, this section can be used as a type of template for integrating ones own custom developed protocols or others provided by third parties (e.g., JSDT [24]).

As shown in Figure 60 and explained above three classes which implement the three main interfaces have to be provided. For LRMP this mapping is:

- LRMP implements Net
- LRMPAddress implements NetAddress
- LRMPDialog implements NetDialog

The class *NetAddress* has to provide implementations for all of the in Listing 25 given methods. This is here straight forward, as basically most methods only return the properties of the given address. Only the actual comparison and generation of the address require some thought. The comparison will get a *NetAddress* object for comparison. Here the first step is here to determine if this other *NetAddress* has the same implementing class (i.e., the same type). If this is the case the relevant protected fields, (which are only known if it has the same implementing class) can be in turn compared. The comparison functionality is as well the key to generating new unused addresses. This is basically an iterative process of creating a new address via some creation schema and checking if it already exists. The currently implemented schema is rather simple as it relies only on changing the port until an unused address is found. The schema as used by the MBone tools to generate Multicast addresses (after all this is what is needed for LRMP) is more sophisticated and should be integrated in the future.

The *LRMPDialog* provides the user with a dialog allowing him to configure a specific address. Therefore an editable field for each of the address characteristics has to be generated. For LRMP these are:

- multicast IP address
- port
- scope defined via a TTL
- a bandwidth limit

An important point is that this dialog is as well responsible for checking that an address is valid (e.g., that the port is in the allowed range). Optional the dialog can be set up to force the user to select a not yet allocated address by checking with the *NetManager* if the address is marked as used.

For the implementation of the *Net* interface the LRMP class is used. This class handles the sending of objects. This includes their serialization and might contain further processing (e.g., segmentation), as is the case for LRMP. Further, the receiving of data is handle by this class, again including the necessary processing. These are in the case of LRMP only serialization and segmentation issues. Next the reconstructed object is handed over to the receiver (i.e., the class implementing *NetReceiver*, as passed during *Net* instantiation). The actually code to do so includes only the segmentation/de-segmentation and the serialization/de-serialization, and, thus, is not complicated.

6.5 Performance of the transaction system

During the design of MACS the issue of supporting efficiently the different actions taking place within the system without compromising data integrity was a key issue. This did lead to the development of the transaction system (see 3.4.3).

The transaction system improved reliability quite substantial (due to guaranteed data integrity) which is especially under heavy load situations clearly noticeable. This section will take a look at the performance of the transaction system based on extensive measurements.

First the measurement scenario is presented, followed by a discussion about the expectations associated with the introduction of the transaction system. Next the measurement techniques are described, followed by a detailed discussion of the measurement results. Finally an evaluation is given.

6.5.1 Measurement scenarios

The main point of interest is the scalability of the system regarding the number of participants in a session. Here the main aspect is the processing of incoming join requests by the chair of the session (compare section 2.4.2.2, especially Figure 9). This is best done using a lecture/class room scenario, as measurements with more than the otherwise set restriction of the number of participants are required. Further an implicit access control is used, as for large scenarios an explicit (thus interactive) access control is not usable due to the resulting manual interactions with the session chair. As this analysis focuses on the scalability of the control mechanisms no applications requiring (large) variable amounts of processing power are executed within the session. For a real usage scenario this would have to be considered. Here an investigation of typical application usage within a session would be of interest, as, for example, an active whiteboard will cause no additional load during the join process of the other participants. It will only cause (a substantial) load once the participants start their own whiteboard, which might well be larger than the load peak caused by the control mechanisms processing the many initial join requests.

6.5.2 Performance expectations for the transaction system

First it is important to define the expectations with respect to the transaction system and the by its introduction affected measurables (that is the introduction of a metric to use for measurements, see as well [63] and [72]) as shown in Table 25.

quantities	**unit**
throughput	number of transactions per second - [#/s]
execution time	milliseconds - [ms] - minimum, average and maximum
idle	percent of time system is idle - [%]
memory	average memory per transaction - [kB]

Table 25: Transaction system performance measures

Throughput is a key aspect for both responsiveness and scalability. Throughput basically is the measurement of the total speed of the system and is strongly related to the execution time of transactions. In the sequential case (i.e., the case without parallel execution, that is without the transaction system or with the transaction system running at level 1, as defined in Table 13) the throughput will simply be the inverse of the average execution time for a transaction. For the parallel case (i.e., the case with the transaction system enabled running at level 3) this will similarly be the inverse of the average execution time multiplied by the average number of simultaneous transactions. Thus, execution time and number of simultaneous transactions become the key measures. It is important to note, that one expects the throughput (speed) of

the system to be slightly reduced by the transaction system, as the processing power available will remain the same, but a certain management overhead is introduced by the transaction system. The only exception possible would be if the sequential case generates waiting conditions which result in an idle processor by blocking other processes. For this scenario this is not likely to occur, as such conditions are mainly caused by actions initiated by the user (via the GUI) and not by incoming requests via the network directed to the session chair.

Thus, as stated, the execution time becomes the main point of interest. The execution time can be subdivided into two parts, the wait time and the run time. The wait time is the time between submitting (SUBMIT) the transaction and the moment it is moved from the input queuer into the running set (START), as explained in section 3.4.5.1. The run time is the time the transaction is actually in the running set, that is the time it takes to process the transaction code. It extends up to the moment the transaction is removed from the running set. A special case are periodic transaction. These will after each execution be transferred to the waiting set and there put to sleep until their next execution interval arrives. Only then they will become eligible for resubmission and hence will be included in the normal scheduling mechanisms. A further point to note is, that the runtime may include possible wait times due to, for example, timeouts on network communication. These times are relevant on how fast a transaction can be processed and, thus, need to be included in the run time. The run time is a straight measurement of a time interval and is thus not directly related to the actual CPU time the transaction did consume.

For the waiting time one expects a significant decrease with parallel execution enabled by the transaction system, as it is in most cases (based on the given dependencies) no longer necessary to wait for other transactions to finish. The run time on the other hand is expected to increase significantly, as the processing power now has to be shared with potentially many other transactions. As stated above the total execution time is likely to increase slightly, as after all no additional processing power is available by the introduction of the transaction system itself. Here it should be noted that this improves with the usage of a server group setup, as then the additional servers actually add real processing power to the system.

The idle time can be divided into two types. The idle time of the transaction system, being the time no transactions are in the running set and the idle time of the complete system, thus being the classical processor idle time. Latter is dependent on a large number of issues and difficult to match to any actions taking place within the transaction system. The transaction system idle time on the other hand is quite relevant, but mainly for the duration of the session, as during peak join times it is to be expected that the transaction system is not idle at all. The aim of the following measurements is exactly to obtain full load measurements, thus the idle time of the transaction system will not be considered any further.

Finally the memory requirements of the transaction system are a possible measurement quantity. This is relevant to determine system requirements for running certain scenarios on a given machine. For the following measurements this is not further investigated, as for the here taken measurements a machine providing sufficient memory was chosen.

6.5.3 Transaction system performance measurement

Before discussing the results obtained from the measurements it is important to have a look on the techniques used to obtain these values in order to consider possible effects on the results caused by the measurements themselves.

6.5.3.1 Measurement techniques

It is difficult to obtain measurements for scalability aspects by monitoring real scenarios [11]. Not only arranging the actual number of machines (tenth to hundreds) taking part in the measurements is problematic, but as well the many side effects caused by such issues as, for example, the network or differences in machines. Thus the approach to simulate a large number of machines was chosen. To achieve this a dedicated thread is started within MACS in order to simulate the reception of a large number of user announcements and join requests for these users. This is done by inserting the corresponding messages into the communication services of a MACS instance, as shown in Figure 61. The rest of the system is therefore not aware of the simulation. This will have some side effects. The actual thread doing the simulation will require processing time which is taken away from the rest of the system. On the other hand the input under this circumstances will be very well conditioned, as only relevant messages are contained and the system has no further work to do. These two effects will counter each other to some extend.

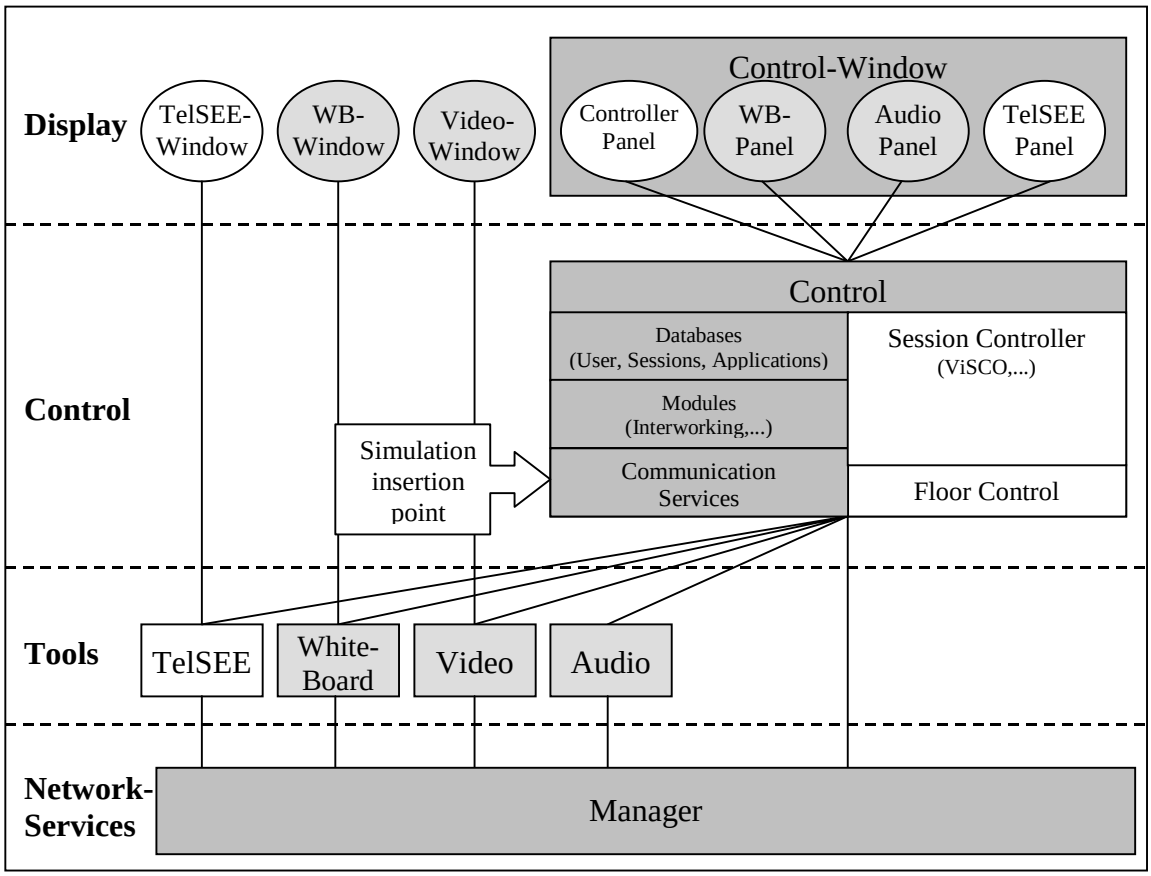


Figure 61: Simulation insertion point

The first set of measurements is done using only ten participants and served, as discussed below, to eliminate other side effects, as, for example, caused by Java class loading. The scalability considerations are based on measurements using 100 simulated participants for a single session. Finally a verification run using 18 real machines[7] (i.e., **no** simulation included) was done.

[7] on Sun Sparc Ultra 1 workstations

6.5.3.2 Results and interpretation

Starting with the above explained simulation of users and join requests for the simple case of 10 participants joining a session (and thus causing the submission of 10 RecvSessionJoin transactions to the transaction system of the session chairs MACS instance) the following is obtained:

	Level1 (sequential)			Level3 (parallel)		
Transaction Name	**wait [ms]**	**run [ms]**	**parTAs**	**wait [ms]**	**run [ms]**	**parTAs**
RecvSessionJoin	46	1532	1	59	1557	9
RecvSessionJoin	1588	7	1	57	1568	8
RecvSessionJoin	1597	8	1	55	1571	7
RecvSessionJoin	1606	8	1	40	1573	6
RecvSessionJoin	1616	8	1	39	1575	5
RecvSessionJoin	1611	8	1	39	1577	4
RecvSessionJoin	1643	8	1	38	1579	3
RecvSessionJoin	1653	10	1	106	1565	2
RecvSessionJoin	1649	9	1	105	1568	1

Table 26: Effect of class loading on meassurements

The values in Table 26 show the sequential execution of level 1 clearly. The long execution time for the first join (i.e., first row) is due to class loading times in Java. Classes are only loaded on demand and this process can be quite time consuming. Further increase in class loading times will occur if the class has to be loaded from a network connected source. For example, starting a locally installed MACS will take about 3 seconds[8], while on the same machine starting MACS from a via Samba mounted drive a total of over 30 seconds is required.

For the level 3 parallel case the same class loading effect can be seen, just that here all transactions are started nearly immediately after submission and therefore all have to wait on the class loading to finish. As a result the setup was modified to execute a single join in advance, wait for the join to conclude and thus guarantee that the class loading did terminate. Only then the measuring process is started.

The variations in the wait and run times arise as the transactions can not be submitted as a block to the transaction system. Thus, the sequential submission of a number of transactions will, first of all, result in different submit times and, secondly, potentially cause a reevaluation of the transactions in the transaction system. Latter may result in a rescheduling of threads within the JVM. This combination of factors makes a continuously increasing or decreasing wait/run time highly unlikely and can even cause substantial variations (e.g., last two wait values in level 3 setup).

[8] on a PC with an AMD K6 450MHz processor

Having removed in the described manner the effect of class loading measurements were taken with 100 simulated participants both for the sequential level 1 and parallel level 3 setup, as shown in Figure 62 and Figure 63 receptively.

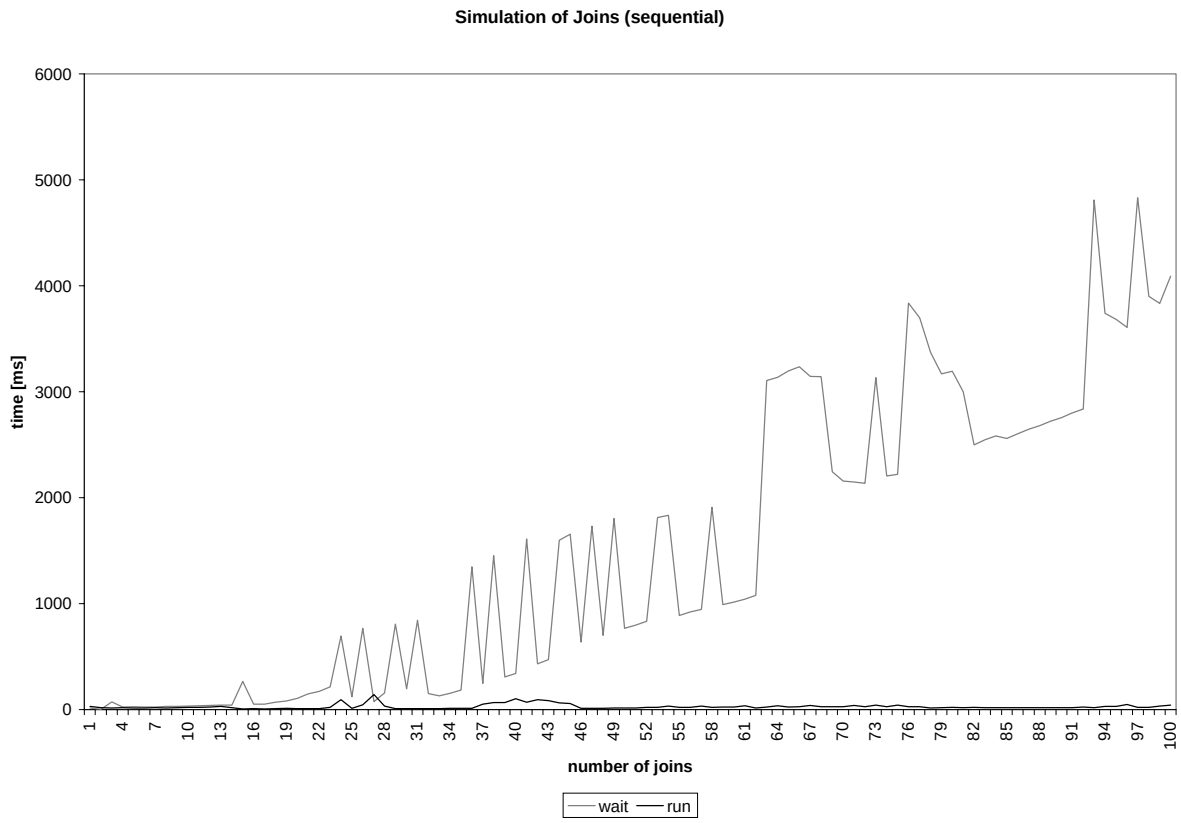


Figure 62: Join simulation for level 1 (sequentiell)

The Figure 62 shows both the wait time and the run time in ms for a simulated 100 join requests. As expected is there little variation in the run time, as only a single transaction runs at a given time and thus processing power does not have to be shared among transactions. Other parts of the system will still have varying processing power requirements causing smaller fluctuations in the run time of the transactions (e.g., the two peaks around the processing of join 25 or the slight increase between join 37 and 45). As well as expected the waiting times will increase with the number of join requests and show a high degree of fluctuation. The latter is dependent on a number of factors such as the submission of the join requests. As the join requests are as well generated by a thread this thread will receive a varying amount of processor time based on the load of the system (Java internal scheduling) this will result in a somewhat bursty submission of join requests to the transaction system. As the transaction system will only allow a single transaction to be executed at each time a reevaluation of the transactions will become necessary after the currently running transaction finishes. This will become with increasing number of transactions in the waiting set a more and more costly process causing additional delays. Other transactions, especially periodic book keeping ones, will as well be entered into the waiting set and cause further delays. The result of these factors can be seen in the above diagram, here at about 15 received join requests the system is no longer able to process the incoming requests directly and thus waiting times start to increase drastically. This is in accordance with the expectations.

Having a look at the same setup but using the transaction system in level 3 the results are shown in Figure 63.

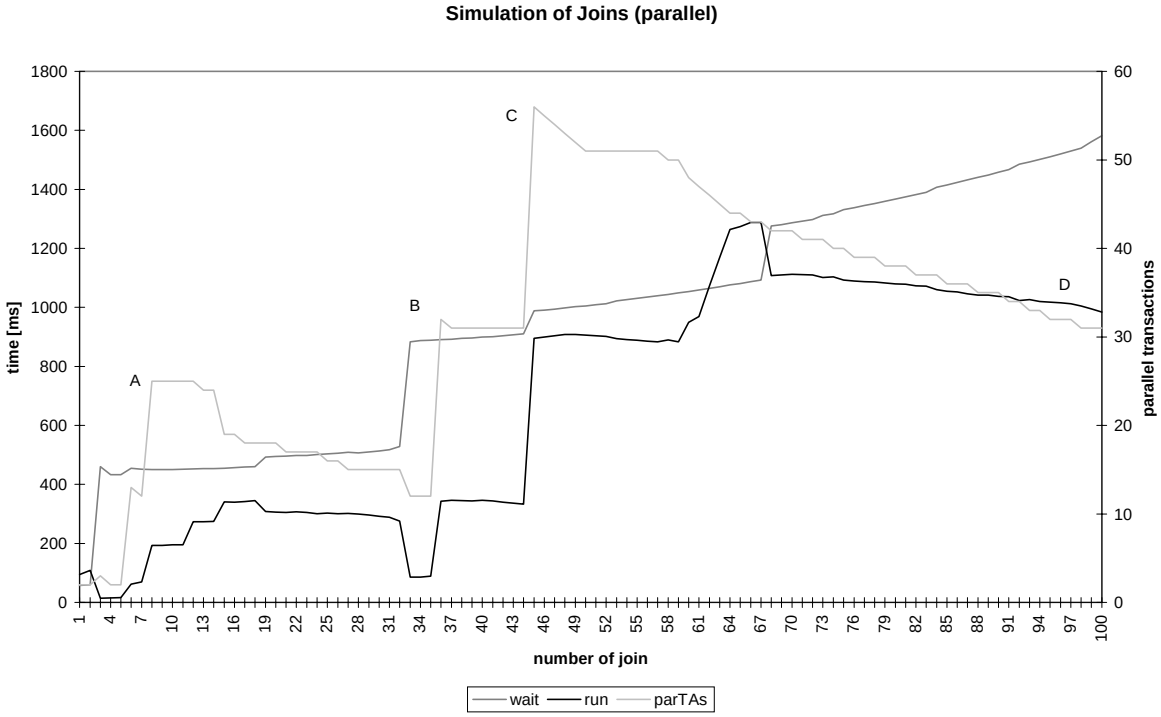


Figure 63: Join simulation for level 3 (parallel)

In Figure 63 not only the wait and run times are shown, but as well the number of transactions being processed by the system at the time the join transaction changed state from SUBMIT to START, that is the moment the transaction was moved from the input queuer into the running set. Having again first a look at the run times it can be seen that these are as expected significantly higher then in the previous case. Further, a correlation between the run times and the number of transactions processed in parallel is clearly visible, an as well expected effect. The before mentioned effect of the transactions being submitted by a thread themselves is here even more clearly identifiable than for the previous case, as here the number of simultaneously executed transactions shows a stair case formation (point A,B,C) on buildup and an afterwards following steady decline (towards point D, which is not 0, as here the number of parallel transactions for the start of the last transaction is given, which is to be expected close to the maximum). With the number of simultaneous transactions the waiting time does as well increase (a clear correlation can here be seen at point B,C), but this much less strongly then in the previous case, an as well expected effect. The load on the system and the transaction system management overhead are responsible for this increase in waiting time, though the here measured values are somewhat higher than expected. Even towards the last transactions the wait time continues to increase. Here one has to note that the system load on removing the last transaction from the waiting set and moving it to the running set is still very high with alone over 30 parallel executing transactions in the running set. The high wait times could possibly be changed by assigning a higher priority to the transaction management thread, but this would require a detailed prior analysis, as higher priority thread can have significant influence on the rest of the system. In general would an investigation of possible prioritization of transactions be interesting, both for the dependency checker as well for the actual transaction execution, as this would provide mechanisms to ensure the fast execution of especially important transactions.

Seeing the expected behavior of run and wait times it is as well necessary to confirm the relation between the number of transactions and the average execution time. This is shown in Figure 64.

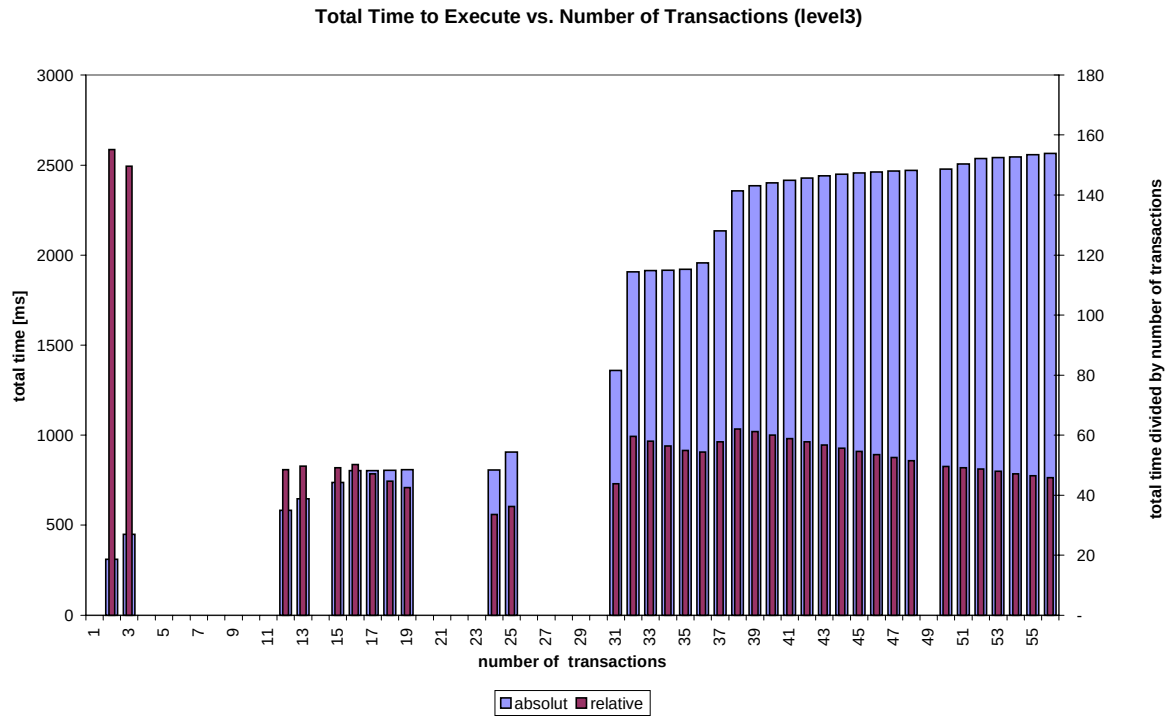


Figure 64: Influence of number of transactions on execution time

Figure 64 shows the expected increase in total execution time for increasing number of parallel executing transactions. Further the calculated equivalent of the single execution time is shown, that is the total execution time divided by the number of parallel running transactions. This is somewhat higher as in the sequential case, which is conform to expectations, as here a management overhead has to be included. The high values for few parallel executing transactions are likely caused by two factors:

- management overhead, which should decrease (due to distribution over multiple transactions) with an increasing number of parallel transactions and
- simulation overhead, which is especially in the beginning caused by the rapid submission of many transactions into the transaction system.

Latter will mainly be noticeable during the initial phase of measurements, as during the execution of the first transactions the tester is still submitting transactions to the transaction system. Especially for the lower half of the graph many open cases (i.e., no measurement for certain number of parallel transactions) exist. The distribution is influenced by the system load and the state of the thread scheduler in the JVM on submission of the transaction. Modifications of the thread priorities of the transactions and the transaction system threads should allow to influence the graph, but this has not been investigated. Basically for the graph

only the upper third shows some identifiable trends, which confirm the decreasing influence of the management overhead by a increasing number of transactions.

Comparing the two sets of measurements, that is comparing the sequential level 1 and parallel level 3 setup, one gets the in Figure 65 shown values. This again confirms expectations in that the sequential case will exhibit greater variations, while the parallel case provides a much smoother curve. Further the parallel case is worse for smaller number of requests, while better for larger numbers. The *breakeven* point is here at about 62 received joins and thus higher than expected. Therefore the transaction system in this simulation only exhibits performance advantages for large number of join requests and the sequential case works better than expected. The in Table 27 shown values confirm this. For level 1 the average wait times are much larger than for level 3, while the average run times are much smaller. The total average time on the other hand is similar with a slight management overhead for the level 3 case. As can be seen this test setup did put a substantial stress on the system resulting in an average number of parallel transactions of 32. Here an investigation of optimal number of parallel transactions would be interesting and could allow to determine an upper limit at which the system performs best.

One has to consider the effects of the test setup. The real scenario is likely to result in better values for the transaction system running in parallel level 3 setup, as it is unlikely that under real circumstances such a concentrated set of uniform (all join) requests will be received. Especially for request needing a substantially longer time to execute or even more significantly forcing in the sequential case due to dependencies transaction system idle times will allow the transaction system in level 3 setup to demonstrate its better usage of system resource despite the added management overhead. Further, the introduction of multi processor setups, as, for example, achieved indirectly via the group of server approach will provide significant advantages, as here a real increase in usable processing power is possible.

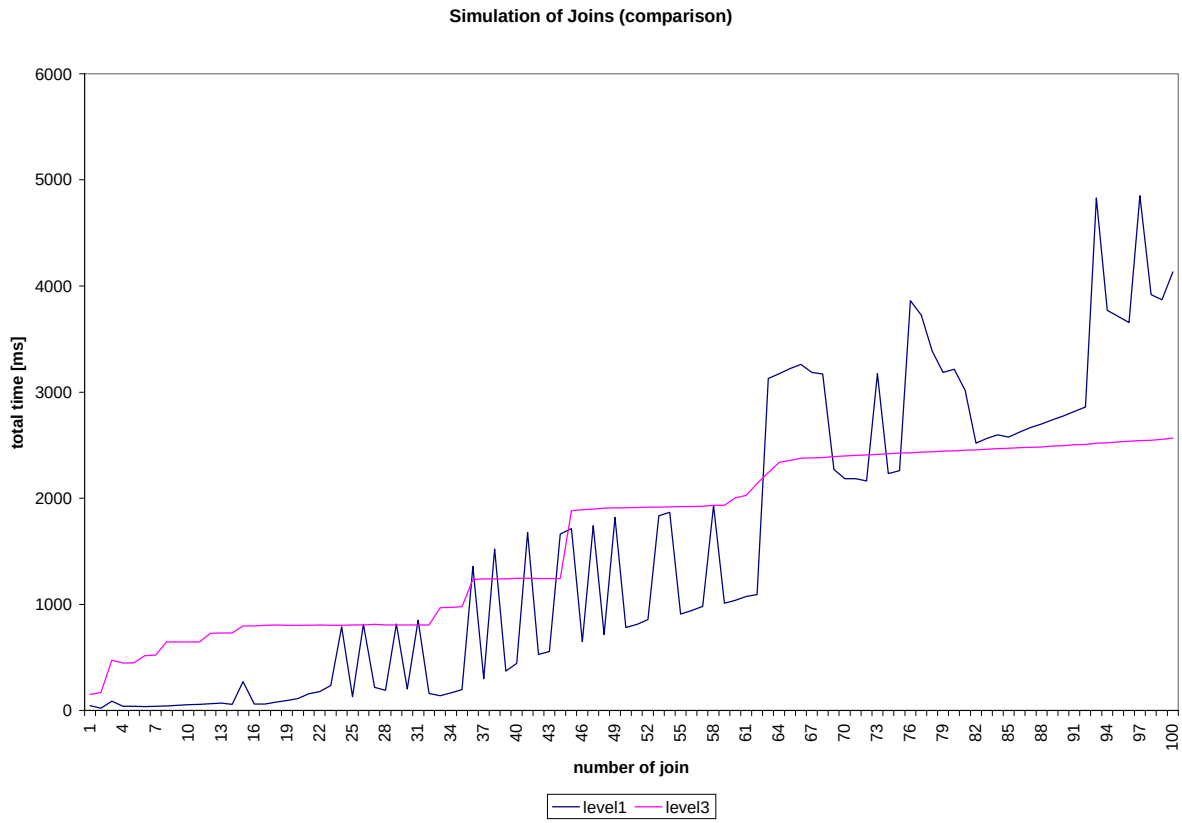


Figure 65: Comparison of simulated joins

Level 1					Level 3			
wait [ms]	run [ms]	total [ms]	parTAs		wait [ms]	run [ms]	total [ms]	parTAs
1523	27	1550	1		955	686	1641	32

Table 27: Comparison of overall average times for simulated joins

Next the results of the simulation will be compared to the values obtained by measurements using a number of (18, as stated above) real machines. These were prepared to automatically join a given session and to leave it again after a short interval (1 sec). This process was repeated continuously. During this test a maximum of 14 participants in the session at the same time occurred, due to the asynchronous joining and leaving of the involved machines.

The two main differences to the simulated case are, first of all, the removed influence of the simulation thread which caused additional load on the system of the session chair and, secondly, the greatly reduced intensity of join request submission, as the join requests are no longer generated in a tight loop and directly submitted, but are now based on messages received from the network. These have first to be decoded and only then a join request is generated. This process causes clearly a less intensive load on the transaction system, as here, as a minimum, the additional time needed by the network level to receive the message and to decode it in order to determine the corresponding transaction will be added to the interval at which transactions are submitted to the system.. This is really an important fact, as with only 18 machines running a join and leave loop for a single session the probability of actually having any significant number of parallel running transactions is very small. This expectation is confirmed by the measurements as shown in Figure 66.

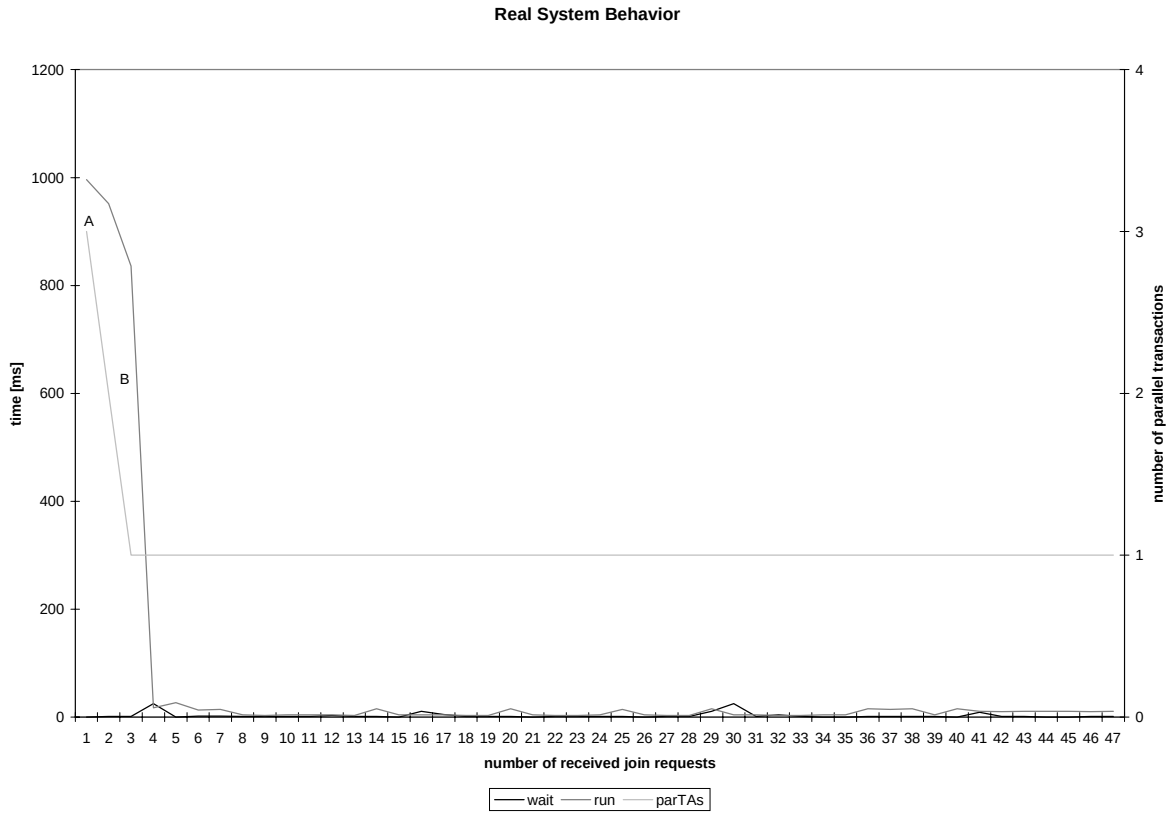


Figure 66: Real system behavior

As can be seen in Figure 66 only two cases with parallel transactions did arise, one with 3 parallel (point A) and one with 2 parallel transactions (point B). As for the automated case a delay was not implemented previously discussed class loading issues did arise causing a substantial run time of the first two transactions. This is exactly the only time simultaneous transactions did occur at all. This supports the hypothesis formulated at the end of the discussion on the simulated results, that the simulations are likely to be rather a harder stress test of the system then probable to occur in real usage scenarios with an equal number of participants. This test should be confirmed with a significantly larger number of machines.

6.5.4 Transaction system evaluation

The measurements basically confirm expectations, in that the transaction system will result in smoother curve for total execution time with smaller wait times but increased run times for the transactions. For small sessions this will not be relevant and, thus, here the transaction system only provides the required de-coupling but yields no performance advantages. For larger session the transaction system will provide better responsiveness due to a more uniform response time despite the small increase in the total average response time. Significant advantages will only be gained once real parallel execution becomes possible, as is the case for the group of server setup.

The values further indicate clearly that MACS control mechanisms will easily be able to handle the discussion group scenario, as here the reduced number of participants causes no significant impact on the overall system load. For the lecture/class room scenario a significant load will be experienced if the maximum number of participants envisaged for this scenario are involved, as shown by the simulation. Still the MACS control mechanisms will be able to

cope, especially as the simulated results are likely to represent worst case values. Under real circumstances the load of the simulation thread will not be present and it is **very** unlikely that requests will be received from basically all participants nearly at the same time. The real scenario should exhibit a much less *aggressive* distribution of join requests.

It is as well apparent that for larger scenarios as the podium discussion additional mechanisms will be necessary. Here an evaluation of the server group setup will be a first step to determine realistically achievable number of users. Further implementation of suggested approaches, as, for example, the hierarchy concept should help to further extend the maximal number of participants making such scenarios feasible.

6.6 Real world experience with MACS

MACS has been submitted constantly to various tests. These were normally executed by the MACS team after each meeting and used to evaluate newly integrated features and to set milestones for the next meeting. These tests are due to the local setup and the choice of testers (i.e., the MACS team) not really representative.

The first real life test using MACS was done in June 2000 in a seminar which took place between the University of Hanover and the Technical University of Braunschweig. The participants had no prior knowledge about MACS and the setup was used to deliver a real seminar. At the University of Hanover two MACS instances with each two students were used, while at the TU-Braunschweig three MACS instances with each two students and a MACS instance for the moderator were used. As a result a total of six instances of MACS catering for 10 students and one moderator were involved in the seminar session. Due to the late availability of the JMF V2.1 just three weeks before the test it was decided to run audio and video via an existing ATM link, as shown in Figure 67. Between the LANs of the two participating institutes a direct MBone tunnel was established, as the normal indirect MBone connection at the time proved to be too unstable. Video was projected by a data projector onto the wall in each of the rooms. This and the handling and mixing of audio was done by a dedicated machine on each side and supervised by a technical assistant. The audio playback was done via speakers for the complete room.

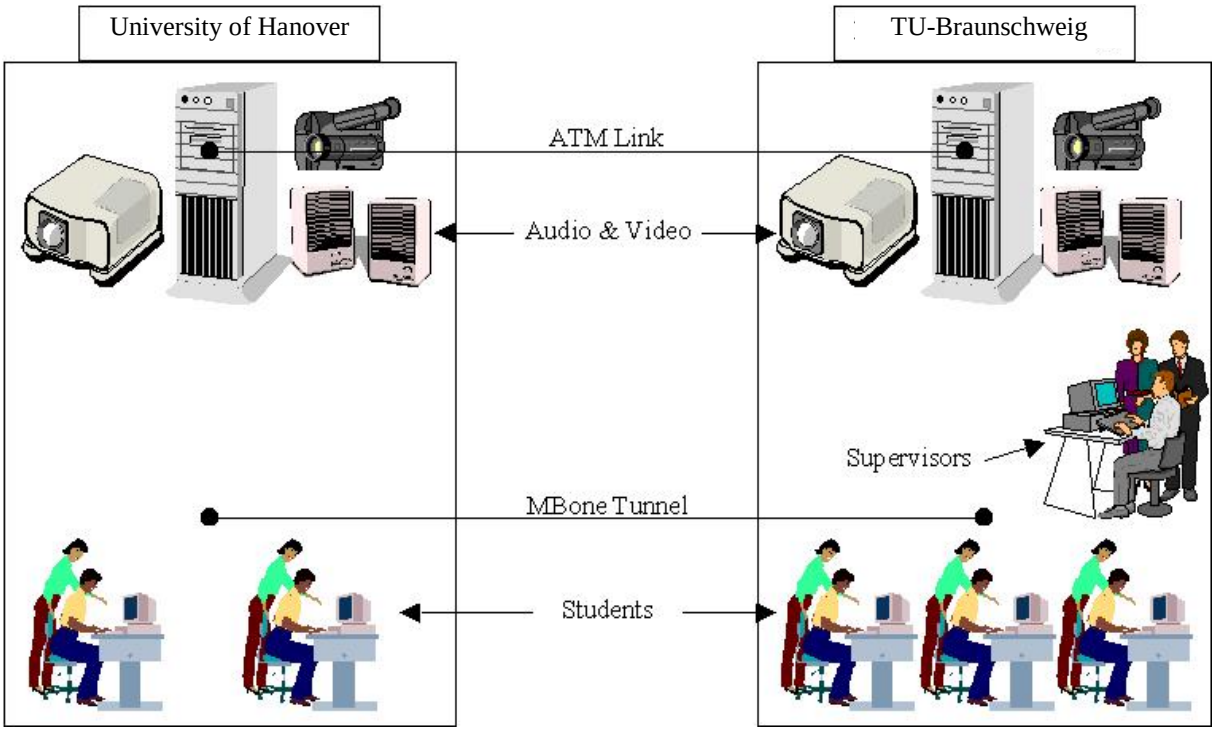


Figure 67: Setup for seminar

The setup did use a discussion group scenario, as defined in section 2.2.1 and shown in Figure 34. Each of the groups did present in turn a talk to the others. For slides the MACS whiteboard was used. Feedback and questions were handled via the session controller. At the end of each session some time for questions was allocated.

Despite the technical limitations and the existing inconsistencies between the original planned discussion group scenario and the scenario resulting from the usage of shared audio and video for each of the rooms, the seminar was perceived well by the students. They did comment though, that individual audio and video feeds and separate rooms would have been more appropriate for the setup. Further it was noted, that they felt less inhibited by the computer than by actually talking in front of a real audience. This is likely an effect associated with their little or no experience with talking in front of an audience. A more complete analysis as well as further details about the setup can be found in [15] (see [35] as a comparison for further not MACCS specific experiences with CSCW systems).

Besides some improvements in tool usage and the better integration of audio and video the actual physical separation of participants is the main aim for the next seminar planned.

7 Conclusion

Based on the introduced motivation, design and implementation of MACS this chapter has first a look at the achieved results and lessons learned in the design and implementation of MACS. Then it has a look at how future developments can progress and what kind of improvements, additional services and "cool" features are possible.

The aim of this work was to develop a portable, flexible and scalable framework for CSCW applications allowing an easy, efficient and fast creation of customized components and thus the creation of a usage scenario specific system.

Portability is a key factor for easy deployment in a heterogeneous environment. The development of MACS used Java 2 as an implementation platform. The overall portability has already been a key factor during development, as MACS was implemented in a heterogeneous environment with different members of the development team working on Solaris, Linux and Windows based systems. The only real problem area hereby is the hardware access as needed for audio/video capture and playback. This problem is in parts solved by JMF, which at the currently available stage of development still exhibits a number of limitations.

Regarding the second key factor, flexibility, the MACS framework excels. Very different applications can easily be integrated into the framework and internal modules allow to extend/supplement the capabilities of the framework. This highly modular concept works well in practice and the configuration via the provided configuration editor is easy to handle. The two level setup of system and user specific configurations allows efficient system adaptation for different deployment environments. Despite its flexibility MACS manages to maintain a tight integration of its components.

The point of scalability of the framework has to be considered in two steps. First the scalability of the control and second of the provided application modules. The control is fairly scalable more so than, for example, ITU based systems. As demonstrated by the measurements even large lecture/class room scenarios can be efficiently supported. Further extensions to even larger scenarios are possible, but may require implementation of some of the marked enhancements, as again summarized in the outlook.

From the application modules presented in this work the media and chat modules are highly scalable and only limited by the underlying network. The whiteboard is still quite scalable, but is somewhat limited and thus will not support very large groups of hundreds of participants. The feedback tool was designed mainly for a class room scenarios and its usability, regarding its concept and not just implementation is for very large groups questionable.

A further issue regarding the scalability of sessions is the availability and support by a suitable session controller module (especially regarding visualization), an issue outside the scope of this work (refer to [19]). The same applies to scalability issues for reliable network protocols supporting group communication.

A further point of interest was the issue of interworking with other CSCW systems. Here the two introduced interworking modules show that it is quite well possible to provide such functionality within the MACS framework in a very flexible way. While interworking has been successfully demonstrated the limitations during interworking mainly caused by using a common (minimal) denominator have as well been highlighted. This limitations have to be considered for interworking scenarios. Here the advantages of interworking versus other

alternatives should be analyzed prior to deployment. Additional scenario specific interworking support features might be needed to support real deployment situations.

The MACS framework as here introduced provides overall an extremely flexible environment for applications and development of customized CSCW systems. It is more portable than other integrated systems. The scalability of MACS exceeds that of many other systems, but certain limitations do not yet allow to serve scenarios requiring massive scalability (as in scenarios with many hundreds or thousands of participants). A number of approaches to improve this have been discussed and some are being implemented in the ongoing development of MACS. Finally basic interworking capabilities with other systems have been provided.

7.1 Outlook

This section takes a look at possible future developments and research areas of MACS and their relevance.

There are three basic types of further developments to be considered, being fine tuning aspects, extensions to existing features, and addition of new features

Considering fine tuning, especially the transaction system offers a wide variety of possibilities. The introduction of adjustable transaction and thread priorities could be used to achieve a per scenario customized management behavior of the transaction system, which could even be dynamically adjusted based on factors such as current number of participants. Similar a mixed mode approach combining dynamic adaptability with predetermined limits could be of advantage for certain setups. An example would be to set a predetermined upper limit on the number of parallel executing transactions in order to prevent excessive system load. Further, an extension to dependency resolution by dividing large transactions into worksteps and checking these for resource dependencies may provide additional improvements. It is as well possible to define cost functions based on obtained measurements in order to reduce possible wait times within the transaction system (due to input queue flushing) as currently needed for restrictive transactions. A further fine tuning target is the system and session information update mechanisms. Improvements can be made allowing to reduce processing and transmission cost. Especially here an adaptation to environment factors, such as network load and total number of detected MACS instances would improve system performance.

Having a look at possible extensions especially the support for massive scalability and, thus, larger scenarios comes to mind. Here the introduced hierarchy concept has to be integrated into all relevant components of the MACS framework including the application modules. This should allow very large conferences. The ongoing implementation of server groups will further increase scalability. Here an improved way to determine an efficient setup for the server group support, as well as the verification of the performance of parallel executed transactions by this server group needs to be investigated. The application support in the MACS framework can as well still be extended. As motivated in the introduction the aim is to make the development of applications easier and provide typical used services through the framework. Here floor control is only a first step. Many applications need late join mechanisms or methods to deal with transitions between asynchronous and synchronous phases of work. These requirements should be supported by services from the framework, thus, helping to reduce the cost of development of such applications while at the same time increasing reliability by component reuse.

For the additions, the most important aim would be the integration of QoS. This has to start with the actual provisioning for such by the networks and will require far reaching support in the framework and by the applications. To achieve this a QoS management needs to be added and integrated into existing mechanisms. Security is a further additional issue which has to be analyzed and integrated into MACS. This not only includes the transmission of data, or the access to sessions, but as well security checks of modules and application loaded at runtime (potentially from a remote server). For this purpose the existing Java security mechanisms should be exploited [77] and their further development closely monitored.

With the increasing availability of powerful multimedia PCs and workstations at the average workplaces CSCW systems will gain more and more relevance and many new usage scenarios will become feasible. The provision of flexible and powerful CSCW frameworks will make the customization for specific usage scenarios and the support of the participants with more powerful applications possible. The integration of computer and telephony, as well as video will finally lead to the widespread usage of multimedia supported, group aware systems based on CSCW frameworks. With further extensions towards mobile systems this will greatly change the simple way we communicate today by use of the telephone to a much more powerful and integrated approach.

A Appendix

A.1 Examples of database entries

A.1.1 Session entry

The session entry contains the following data about a session:

```
ObjectID id;            // the ID of the session
ObjectID creatorID;     // the ID of the session creator
String name;            // the session name
String email;           // the email contact for this session
String url;             // a URL with more info about the session
String description;     // a description of the session
String type;            // the type of the session (see usage scenario)
NetAddress group;       // the communication group used for session control
boolean explicit;       // the access mode for the session
List users;             // the list of users
boolean ready;          // the state of the session
long timestamp;         // time the session data was last updated
String controllerClass; // class name of session  controller
MacsSessionController controller;        // the actual session controller
SessionNet snet;        // the actual SessionNet
FloorControl floor;     // the actual floor control
SessionApplicationManager applications;  // the actual application manager
```

A.1.2 User entry

The user entry contains the following data about a user:

```
ObjectID id;            // the ID of the user
String givenName;       // the given/first name of the user
String surName;         // the users last name
String location;        // the user location (e.g., the town)
String country;         // the users country
String organisation;    // the organization the user belongs to
String tele;            // the users telephone number
String email;           // the users email
String url;             // the users homepage
String login;           // the used login
InetAddress host;       // the users host address
long timestamp;         // the time the user entry was last updated
long delta;             // the max time between updates before an expire
Color color;            // the color for this entry in the user list
byte[] image;           // the user thumbnail
Vector hardware;        // the (media) hardware available to this user
Vector applications;    // the applications available to this user
Vector token;           // the tokens owned by this user
```

A.1.3 Application entry

The application entry contains the following data about an application:

```
ApplicationID id;            // the application ID
MacsApplication application; // the actual application class
```

Detailed information about the application can be extracted from the application class via the *MACSApplication* interface.

A.2 Transaction system

A.2.1 List of transactions

List of existing transactions:

- **SetupMacs**
- **ShutdownMacs**
- **EnterMacs**
- **LeaveMacs**
- **SystemAlive**
- **SystemUpdate**
- **RecvSystemIndication**
- **ExpireUsers**
- **CreateSession**
- **RecvSessionCreate**
- **TerminateSession**
- **RecvSessionTerminate**
- **SessionAlive**
- **RecvSessionIndication**
- **ExpireSessions**
- **JoinSession**
- **RecvSessionJoin**
- **LeaveSession**
- **RecvSessionLeave**
- **InviteSession**
- **RecvInviteSession**
- **ExpelSession**
- **RecvExpelSession**

A.2.2 Transaction interdependencies

Below the default configuration file for the MACS transaction system in a parallel level 3 setup is given. For each transaction the group (none, groupnone or list) is specified. If required (i.e., for group list) a list of dependencies is supplied. Via the log flag the logging for

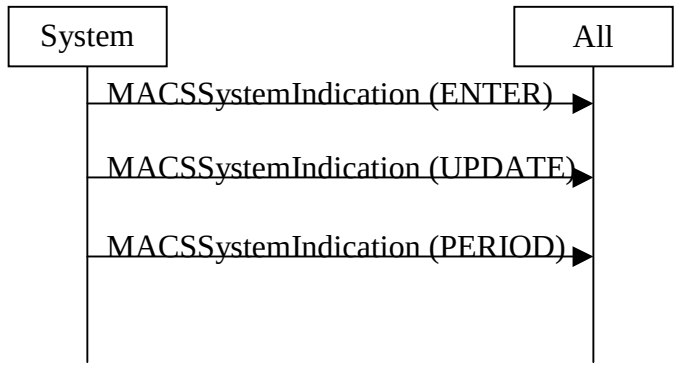
each transaction can be activated. This is useful for taking measurements, especially if used together with the tester module.

```
SetupMacs=none
SetupMacs.log=true

EnterMacs=none
EnterMacs.log=true


LeaveMacs=none
LeaveMacs.log=true

ShutdownMacs=none
ShutdownMacs.log=true

ExpireUsers=none
ExpireUsers.log=true

SystemAlive.list=SetupMacs EnterMacs SystemAlive
SystemAlive.log=true

SystemUpdate.list=SetupMacs EnterMacs SystemUpdate
SystemUpdate.log=true

RecvSystemEnter=none
RecvSystemEnter.log=true

RecvSystemIndication=none
RecvSystemIndication.log=true

CreateSession=groupnone
CreateSession.log=true

SessionAlive=groupnone
SessionAlive.log=true

ExpireSessions=none
ExpireSessions.log=true

RecvSessionJoin=list
RecvSessionJoin.list=SystemEnter SystemLeave RecvSystemIndication
ExpireUsers SessionCreate SessionTerminate SessionJoin SessionLeave
SessionAlive RecvSessionCreate RecvSessionTerminate RedvSessionInvite
RecvSessionExpel RecvSessionIndication ExpireSessions
RecvSessionJoin.log=true
```

```
RecvSessionLeave=list
RecvSessionLeave.list=SystemEnter SystemLeave RecvSystemIndication
ExpireUsers SessionCreate SessionTerminate SessionJoin SessionLeave
SessionAlive RecvSessionCreate RecvSessionTerminate RedvSessionInvite
RecvSessionExpel RecvSessionIndication ExpireSessions
RecvSessionLeave.log=true

RecvSessionCreate=groupnone
RecvSessionCreate.log=true

RecvSessionTerminate=groupnone
RecvSessionTerminate.log=true

RecvSessionInvite=groupnone
RecvSessionInvite.log=true

RecvSessionExpel=groupnone
RecvSessionExpel.log=true

RecvSessionIndication=groupnone
RecvSessionIndication.log=true
```

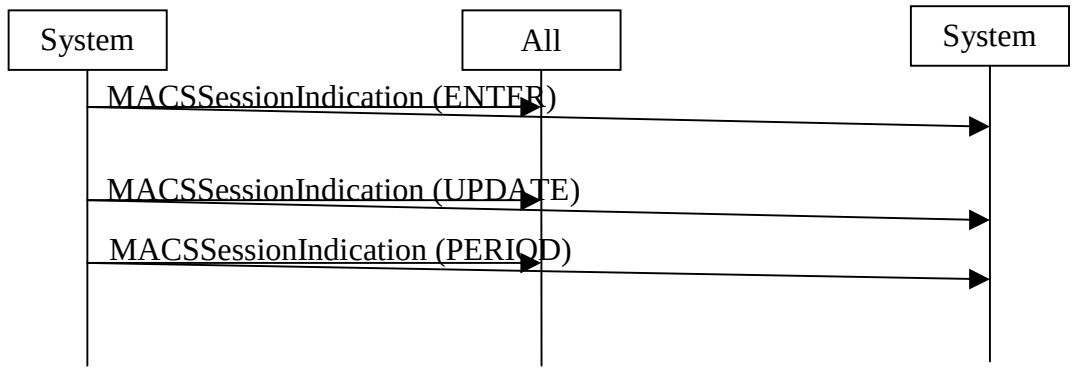
A.3 Messages of control protocol

A.3.1 System messages

<table>
<tr><td colspan="3">Name: MACSSystemIndication</td></tr>
<tr><td colspan="3">Description: This message is used to send information about the local system to other systems:
• as an announcement when starting the local system (type ENTER)
• as an update for changes of the information about the local system (type UPDATE)
• as a periodic alive message (type PERIOD)</td></tr>
<tr><td colspan="2">Channel: global</td><td>Type: system</td></tr>
<tr><td colspan="3">Parameter</td></tr>
<tr><td>• sequence</td><td colspan="2">sequence number</td></tr>
<tr><td>• state</td><td colspan="2">system status (ENTER, UPDATE, PERIOD)</td></tr>
<tr><td>• user</td><td colspan="2">description of user</td></tr>
<tr><td colspan="3">Diagram
 
</td></tr>
</table>

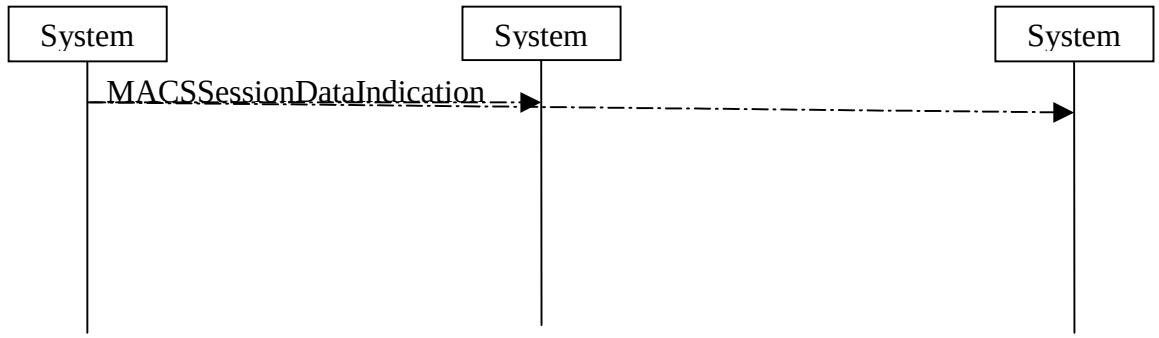
A.3.2 Session messages

Name: MACSSessionIndication	
Description: This message is used to send information about a session to other systems: • as an announcement after starting the session (ENTER) • as a periodic alive message (send by system which created the session) (PERIOD) • as an update if session relevant information has changed (UPDATE)	
Channel: global	**Type:** session
Parameter	
• sender	identify the sender
• sequence	sequence number
• state	session status (ENTER, UPDATE, PERIOD)
• session	session description
Diagram	



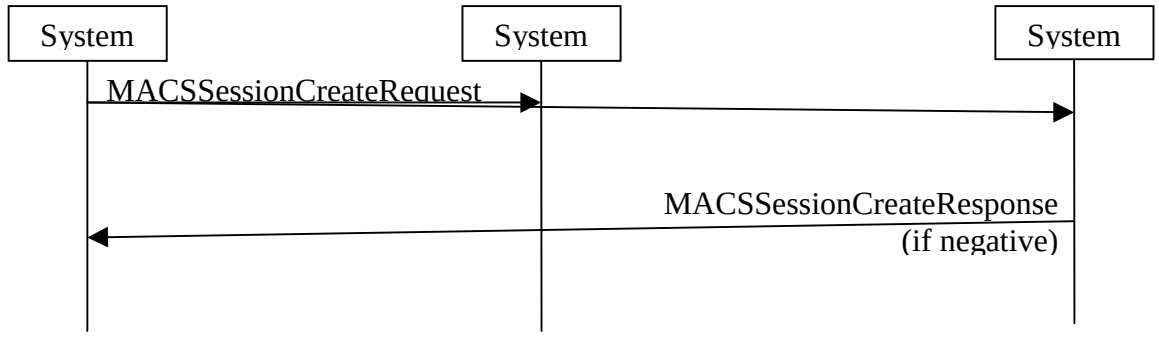
Name: MACSSessionDataIndication	
Description: This message is used to transmit data between different session controllers.	
Channel: session	**Type:** session
Parameter	
• sender	identify sender
• sequence	sequence number
• data	the data

Diagram



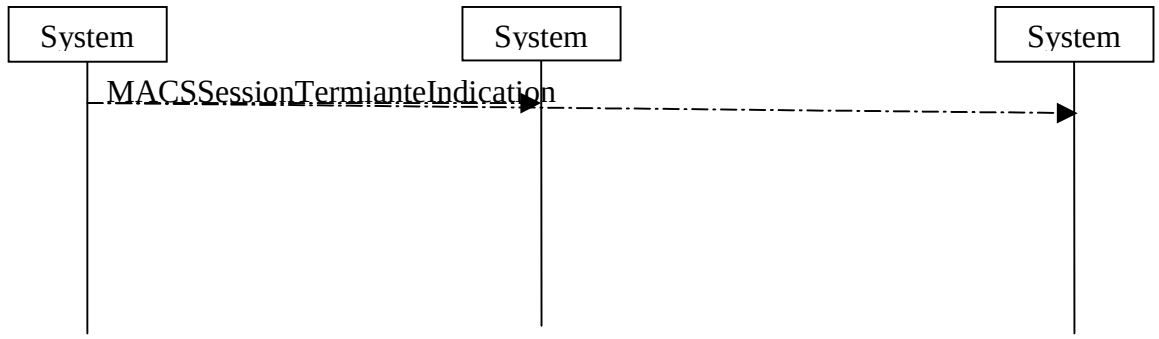
Name: MACSSessionCreateRequest, MACSSessionCreateResponse	
Description: This messages are used to create a new session. Other systems can reply to the create within a given timeout to prevent the creation of the described session (e.g., to prevent naming conflicts).	
Channel: global	**Type:** session
Parameter	
Name: MACSSessionCreateRequest	
• sender	identify sender
• sequence	sequence number
• session	session description
Name: MACSSessionCreateResponse	
• sequence	sequence number
• request	identify sequence number of request
• requester	identify sender of request
• reason	the reason why the session should not be created

Diagram



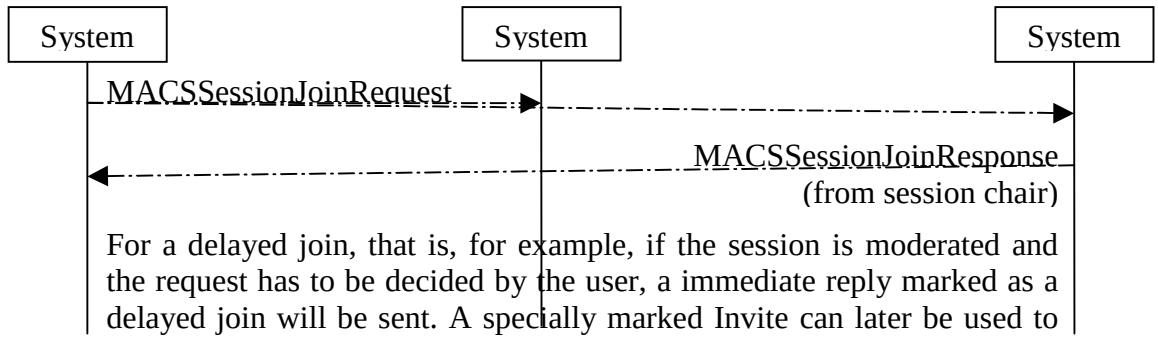
Name: MACSSessionTerminateIndication	
Description: This message is used to indicate the termination of this session.	
Channel: session	**Type:** session
Parameter	
• sender	identifies the sender
• sequence	sequence number

Diagram



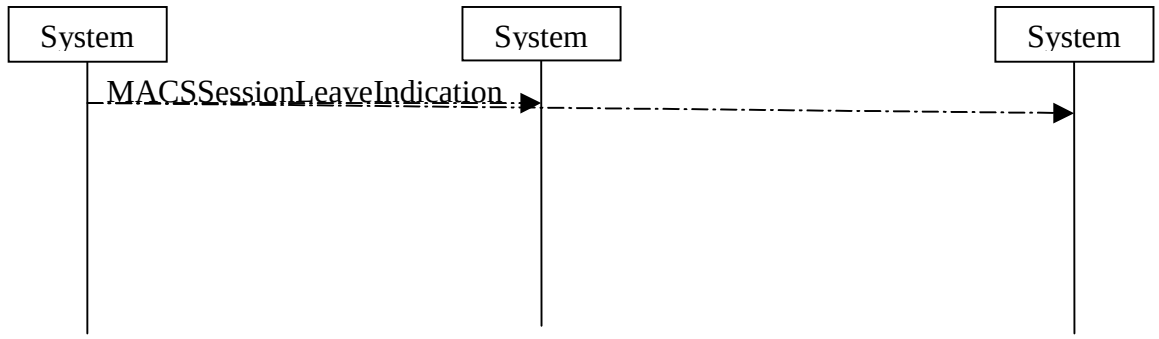
Name: MACSSessionJoinRequest, MACSSessionJoinResponse	
Description: This message is used to join an existing session. The remote control will check for required resources and if these are available pass the request on to the remote session controller.	
Channel: session	**Type:** session
Parameter	
Name: MACSSessionJoinRequest	
• sender	describes the sender
• sequence	sequence number
• request	sequence number of earlier invite request, if applicable (else 0)
• user_data	additional user data (optional, e.g. for Visco)
Name: MACSSessionJoinResponse	
• sequence	sequence number
• request	identify sequence number of request
• requester	identify sender of request
• reason	reason code (OK, or an error code)
• user_data	additional user data (optional, e.g. for Visco)

Diagram



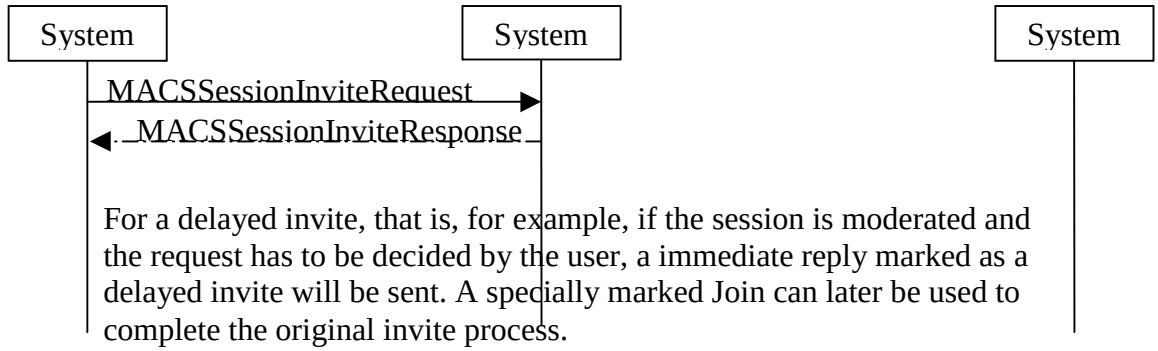
For a delayed join, that is, for example, if the session is moderated and the request has to be decided by the user, a immediate reply marked as a delayed join will be sent. A specially marked Invite can later be used to complete the original join process.

Name: MACSSessionLeaveIndication	
Description: This message is used to indicate the given user is leaving the session.	
Channel: session	**Type:** session
Parameter	
• sender	identify the sender
• sequence	sequence number
Diagram	



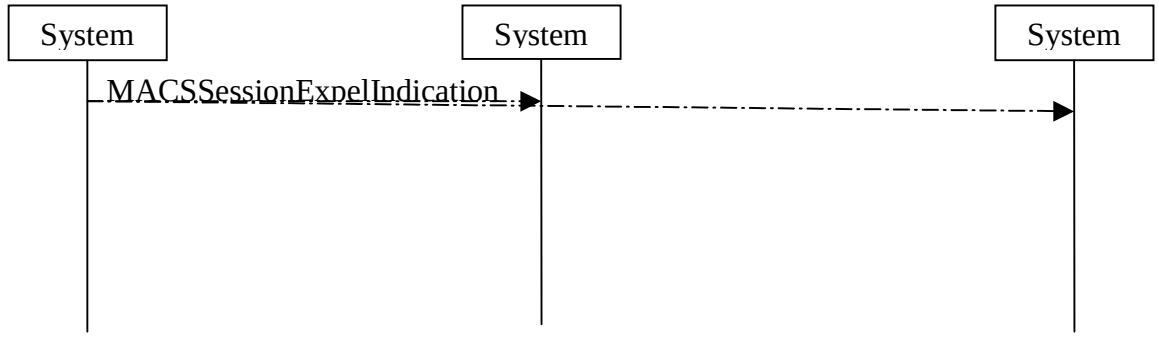
Name: MACSSessionInviteRequest, MACSSessionInviteResponse	
Description: These messages are used to invite a new user to an existing session.	
Channel: global	**Type:** session
Parameter	
Name: MACSSessionInviteRequest	
• sender	identify sender
• sequence	sequence number
• request	sequence number of a MACSJoinRequest, only if this is the result of such a delayed join
• target	identify the person to be invited
• session	identify the session
• user_data	additional user data (optional, e.g. for Visco)
Name: MACSSessionInviteResponse	
• sequence	sequence number
• request	sequence number of request
• requester	identify sender of request
• reason	reason code (OK, or an error code)
• user_data	additional user data (optional, e.g. for Visco)

Diagram



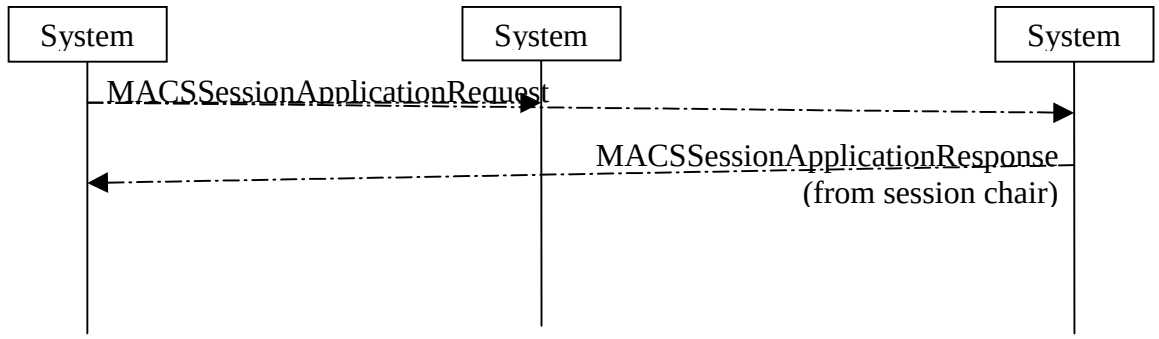
Name: MACSSessionExpelIndication	
Description: This message is used by the session chair to expel a user.	
Channel: session	**Type:** session
Parameter	
• sender	identify sender
• sequence	sequence number
• target	identify user to be expelled

Diagram



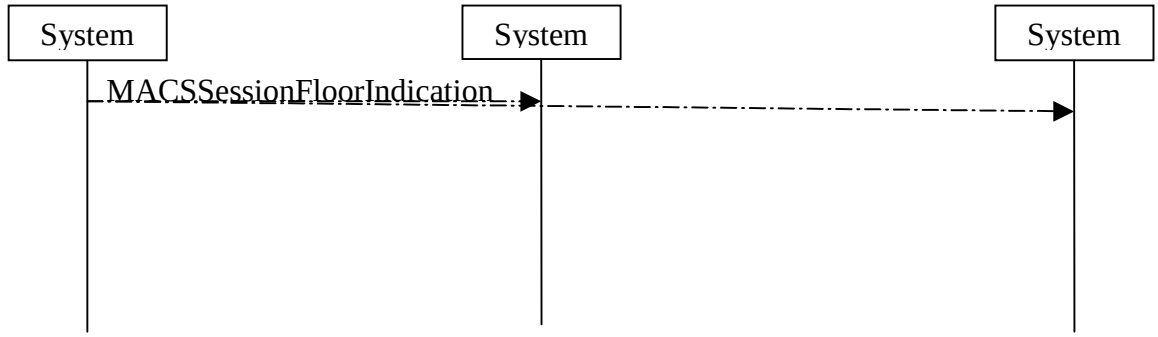
Name: MACSSessionApplicationRequest, MACSSessionApplicationResponse	
Description: This message is used to coordinate the start of an application in a session, especially the assignment of an address for the application to use.	
Channel: session	**Type:** session
Parameter	
Name: MACSSessionApplicationRequest	
• sender	describes the sender
• application	describes the application
• sequence	sequence number
Name: MACSSessionApplicationResponse	
• sequence	sequence number
• request	identify sequence number of request
• address	address assigned

Diagram



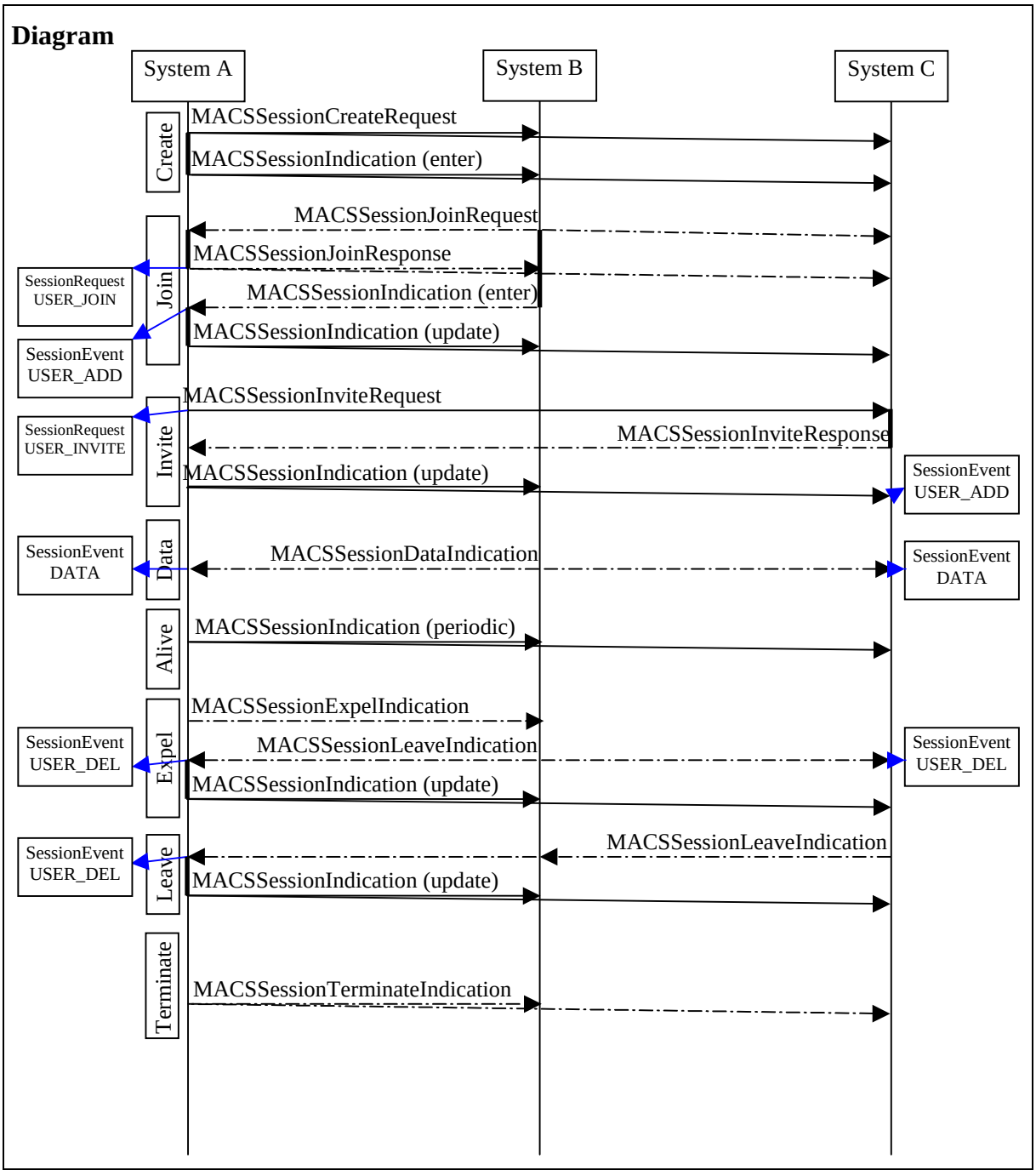
Name: MACSSessionFloorIndication	
Description: This message is used to transmit floor data between different controls.	
Channel: session	**Type:** session
• sender	identify sender
• sequence	sequence number
• data	the data

Diagram



A.3.3 An example of a message sequence for a simple session

This example shows a discussion group in an open session with three participants (A, B and C). User A creates the session. User B joins the session once it is created, but is expelled later by user A, before A invites user C. C leaves the session just before A actually terminates the session.



A.4 Messages implemented by ITU interworking module

Below a list of messages processed by the ITU interworking module is given

A.4.1 Q.931/932 messages

- **Alerting** Informs the calling terminal that the called user is being alerted.
- **CallProceeding** Informs the calling terminal that the call is being processed. Allows to, for example, prevent timeouts for calls via multiple intermediate systems.
- **Connect** Response to setup messages. Confirms that the call was accepted and contains setup details for H.245 control channel
- **Setup** Initial call setup message.
- **ReleaseComplete** Indicates that the control channel has been released and should be closed.
- **Status** This is or a reply to a StatusInquiry or a indication that a message was not understood (e.g., due to not implemented functionality).
- **StatusInquiry** Request actual status.

A.4.2 H.245 messages

- **MasterSlaveDetermination** This set of messages (incl. Acknowledge, Reject and Release) is used in the process of determining the master and the slave for a new connection.
- **CapabilitySet** This set of messages (incl. Acknowledge, Reject and Release) is used to transmit the capabilities of the terminals.
- **OpenLogicalChannel** This set of messages (incl. Acknowledge, Reject and Confirm) is used to establish a logical channel.
- **CloseLogicalChannel** This set of messages (incl. Acknowledge) is used to close an establish logical channel.
- **RequestChannelClose** This set of messages (incl. Acknowledge, Reject and Confirm) is used to request that a specific logical channel between two points is closed.
- **RequestMode** This set of messages (incl. Acknowledge, Reject and Confirm) is used to request a specific transmission mode.
- **RoundTripDelay** This set of messages (Request and Response) is used to determine the round trip time between two nodes and can be used to check if the remote node is still alive.
- **MaintananceLoop** This set of messages (Request, Acknowledge, Reject and CmdOFF) is used to set specific loop back behavior.

A.4.3 H.245 indications

- **Function Not Supported**
- **Logical Channel Active**
- **Logical Channel Inactive**
- **Multipoint Conference**
- **Cancel Multipoint Conference**
- **Multipoint Zero Comm**
- **Cancel Multipoint Zero Comm**
- **Multipoint Secondary Status**
- **Cancel Multipoint Secondary Status**
- **Video Indicate Active To Activate**
- **Video Temporal Spatial Trade Off**
- **SBE Number**
- **Terminal Number Assign**
- **Terminal Joined Conference**
- **Terminal Left Conference**
- **Seen By At Least One Other**
- **Cancel Seen By At Least One Other**
- **Seen By All**
- **Cancel Seen By All**
- **Terminal You Are Seeing**
- **Request For Floor**
- **Jitter**
- **User Input**

A.5 Configuration file for the configuration editor

The extract of the configuration file for the configuration editor shows how preference keys are mapped to configuration editor sections and fields, and, thus, how easily new modules with their corresponding sections and fields can be added.

```
title "MACS-Properties"

category "Appearance"
{
  string "Title"  "control.title"
  slider "Width"  "control.width"  300 1000
  slider "Height" "control.height" 150 800
}

category "Identity"
{
  string "Name"         "user.name"
  string "Given name"   "user.givenname"
  string "Surname"      "user.surname"
  string "Location"     "user.location"
  string "Country"      "user.country"
  string "Phone"        "user.tele"
  string "EMail"        "user.email"
  string "URL"          "user.url"
  string "Organisation" "user.organisation"
  string "Image"        "user.image"
}

category "Applications"
{
  text "Add application-names here and set its class in 'All'-category
  list "control.applications"
}

category "Internet Locator Server"
{
  checkbox "Use ILS service"    "Ils.use"
  string "Server URL"           "Ils.providerURL"
  string "LDAP Version"         "Ils.ldapVersion"
  string "Base object"          "Ils.baseDN"
  string "Search filter"        "Ils.filter"
  string "Category"             "Ils.category"
  slider "Update Interval [ms]" "Ils.updateInterval" 1000 600000
}
```

```
category "ITU Gateway Module"
{
  checkbox "Use ITU-Gateway"    "Itu.use"
  string "Call-Signalling-Port"    "Itu.callport"
}

category "MBone Module"
{
  checkbox "Use MBone Module"    "mbone.use"
}

category "All"
{
  all "" ""
}
```

B Glossary

- **Application**: An application is a program created to perform a specific task. This can, for example, be a drawing application to draw or a text application to edit text. Normally applications only support a single user.
- **Application-Module**: An application module is a module with the characteristics of an application. While it still serves a specific task it is no longer a single entity, but one of many entities in a system. A CSCW system therefore will use a number of application modules to provide content specific capabilities to the user. In a way the CSCW system is therefore an application with the specific task to provide communication/collaboration capabilities to the user.
- **Closely/loosely coupled systems**: A closely or tightly coupled system refers to a setup in which each coupled system has detailed knowledge of the other systems and their states. The advantages of such a setup is that each system can provide detailed information to the user. The states of all systems are known, thus, resulting in an always up-to-date and complete overview. Loosely coupled systems known much less about each other and often do not know all other systems. They may have incomplete or outdated state information about these. The advantages are a greatly reduced management and update overhead resulting in better performance and scalability.
- **Concast**: Concast communication describes the situation were to many transmissions from *m* senders are all directed to one single receiver, as, for example, used for automatic data collection (e.g., distributed weather stations).
- **CSCW**: A Computer system Supporting Collaborative Work, that is, a system allowing people to interact and collaborate via a computer in order to get a certain job (work) done. This is a very general definition and covers a vast variety of systems.
- **Floor**: The floor represents the right/permission to do a specific task with/on a certain resource. Only the application module who defined that resource will be able to interpret the floor and accordingly allow or prohibit the requested operation.
- **Framework**: A framework integrates a number of components into a (more) complex system. Design of components for a framework is normally easier, as certain services are offered by the framework, making the development more efficient.
- **Instance**: Each time a system is started it will be revered to as an instance. There can be more than one instance of a given system on a given host at the same time.
- **MBone**: The MBone is a virtual overlay network covering parts of the Internet. It allows the transmission of multicast data within this network and therefore provides an efficient support for broad-/multicasting information.
- **Multicast**: Multicast communication refers to a transmission from one sender to *n* receivers, as, for example, is the case if a well defined number of users follow the transmission of an event (e.g., a talk or football game).
- **Multipoint**: Multipoint refers to communication between *m* senders and *n* receivers, were typically an overlap between senders and receivers exist. An example for this are typically CSCW systems.

- **Participant**: A participant is a user who has joined a specific session and therefore has become a participant within that session.
- **QoS**: A system/network supporting Quality of Service allows the user to request certain quality guarantees. These can, for example, refer to the bandwidth, delay or loss rate. Due to the quality guarantee the user can determine in advance the behavior of the system (e.g., regarding video quality).
- **Resource**: A resource is an object needed to perform a task at hand. This can be abstract such as a slide in a drawing application module or CPU time. It can as well be a real object, such as a microphones or a video camera. Resources are defined and interpreted by the application module using them.
- **Scalability**: Scalability refers to the ability to increase a certain aspect in number or size. In a CSCW context the two main points of interest regarding scalability are the increase in number of users/participants and the increase in the geographical size/distribution of a CSCW system.
- **Session**: A session forms the frame for a number of users to communicate or collaborate. It isolates the group of participating users from the rest of all users and gives them a common frame. This applies not only to the users, but as well to the applications used by them or more general any resource managed within the session.
- **System**: System can refer to a certain software package installed on a computer, such as a CSCW system, or to the computer/host system itself.
- **Tool**: A tool in the context of a CSCW system is one of its components (e.g., an application module) which is used together with the parts of the CSCW system in order to work on a specific task.
- **Transaction**: A transaction is a job submitted to the system for processing. A transaction has at least one work step, but may be made up out of many. Each work step has certain requirements that need to be fulfilled before it can be executed. These normally refer to the access to specific system resources. Once all work steps have been executed the transaction is concluded and thus the result of the transaction is returned.
- **Unicast**: Unicast communication refers to a direct transmission of one sender to one receiver, as, for example, used in the communication between two users.
- **User**: A user is the person using a certain instance of a CSCW system. Normally a instance is coupled to exactly one user. The opposite is not necessarily true, that is that each user is only using one instance of a CSCW system at a given time, though there are few reasons to run more than one instance per user.

C Abbreviations

AOL	America Online
API	Application Programming Interface
BSCW	Basic Support for Cooperative Work
CAD	Computer Aided Design
CASE	Computer Aided Software Engineering
CSCW	Computer Supported Collaborative Work
GCC	General Conference Control
GIF	Graphic Interchange Format
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ILS	Internet Location Server
IPC	Inter Process Communication
IRC	Internet Relay Chat
IRI	Interactive Remote Instruction
IT	Information Technologies
ITU	International Telecommunication Union
JDK	Java Development Kit
JETS	A Java-Enabled Tele-Collaboration System
JIT	Just In Time (e.g., compiler)
JMF	Java Media Framework
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LRMP	Lightweight Reliable Multicast Protocol
MACS	Multicast based Advanced Conferencing System
MCS	Multipoint Communication Services
MCU	Multipoint Communication Unit
PDA	Personal Digital Assistant
QoS	Quality of Service
RFC	Request For Comment
RTP	Real Time Protocol

SAP	Session Announcement Protocol
SCCP	Simple Conference Control Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
TelSEE	Tele Software Engineering Editor
TTL	Time To Life
UML	Universal Modeling Language
VRML	Virtual Reality Meta Language

D References

[1] Abdel-Wahab H., Favereau J., Kabore P., Kim O., *Issues in Platform-Independent Support for Multimedia Desktop Conferencing and Application Sharing*, IFIP Conference on High Performance Networking (HPN'97), White Plains, NY, USA, 04+05/1997

[2] Attie P., Rusinkiewicz M., Sheth A., Singh M., *Specifying and Enforcing Intertask Dependencies*, 19th VLDB, Dublin, Ireland, 1993

[3] Bapat S., *Object-Oriented Networks*, Prentice Hall, New Jersy, USA, 1994

[4] Beca L., Cheng G., Fox G., Jurga T., Olszewski K., Podgorny M., Sokolowski P., Stachowiak T., Walczak K., *Tango - a collaborative environment for the world-wide web*, Technical report, Northeast Parallel Architectures Center, Syracuse University, USA, 1999

[5] Beca L., Cheng G., Fox G., Jurga T., Olszewski K., Podgorny M., Sokolwski P., Walczak K., *Web Technologies for Collaborative Visualization and Simulation*, Technical Report SCCS-786, Northeast Parallel Architectures Center, Syracuse University, USA, 01/1997

[6] Beck-Wilson J., Pfister H., Schuckmann C., Wessner M., *The Metaphor of Virual Rooms in the Cooperative Learning Environment CLear*, CoBuild, Darmstadt, Germany, 1998

[7] Bentley R., Horstman T., Trevor J., *The World Wide Web as enabling technology for CSCW: The case of BSCW*, Computer-Supported Cooperative Work, Special issue on CSCW and the Web, Vol. 6, 1997

[8] Birman K., Maffeis S., Renesse R., Horus R., *A Flexible Group Communication System*, ACM, Communications of, Vol. 39, No. 4, 04/1996

[9] Böger A., Brand O., Harbaum T., Sturzebecher D., Zitterbart M., *EduNet: Internet-based Education Network*, 2nd Conference on New Learning Technologies, Bern, Switzerland, 08/1999

[10] Böger A., Brand O., Harbaum T., Zitterbart M., *Teaching Computer Networks across Networks*, EDEN, Bologna, Italy, 06/1998

[11] Borghoff U., Schlichter J., *Rechnergestützte Gruppenarbeit - Eine Einführung in Verteilte Anwendungen*, Springer Verlag, Berlin, Germany, 1995

[12] Bormann C., Ott J., Reichert C., *Internet Draft - Simple Conference Control Protocol*, IETF, MMUSIC, 06/1996

[13] Brand O., Mahalek W., Sturzebecher D., Zitterbart M., *MACS - Eine modulare Kollaborationsumgebung*, Praxis der Informationsverarbeitung und Kommunikation (PIK), Saur Verlag, Munich, Germany, 04/1999

[14] Brand O., Mahalek W., Sturzebecher D., Zitterbart M., *MACS: A flexible and scalable collaboration environment*, ED-MEDIA, Freiburg, Germany, 06/1998

[15] Brand O., Petrak L., Sturzebecher D., Zitterbart M., *Bericht über das Tele-Seminar zwischen der Universität Hannover und der TU Brausnchweig*, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 07/2000

[16] Brand O., Petrak L., Sturzebecher D., Zitterbart M., *Supporting Tele-Teaching: Visualization Aspects*, 26th Euromicro Conference, Maastricht, Netherlands, 09/2000

[17] Brand O., Sturzebecher D., Zitterbart M., *Distance Learning with MACS*, ED-MEDIA, Seattle, USA, 06/1999

[18] Brand O., Sturzebecher D., Zitterbart M., *MACS - A Modular Collaboration Environment*, Internet and Multimedia Systems and Applications (IMSA '99), Nassau, Bahamas, 10/1999

[19] Brand O., Visualisierung von Sitzungskontrolle in CSCW Systemen, doctoral thesis, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, in preparation

[20] Brand O., Zitterbart M., *Steuerung von Konferenz- und Kollaborations-Anwendungen*, Praxis der Informationsverarbeitung und Kommunikation (PIK), Saur Verlag, Munich, Germany, 04/1997

[21] Brand O., Zitterbart M., *Visualization of session and floor control in the collaboration system MACS*, HCI 99, Munich, Germany, 08/1999

[22] Buford J., *Mutlimedia Systems*, Addison-Wesley, New York, USA, 1994

[23] Burger C., *Groupware - Kooperationsunterstützung für verteilte Anwendungen*, dpunkt, Heidelberg, Germany, 1997

[24] Burridge R., *Java Shared Data Toolkit User Guide*, Sun Microsystems, online, 10/1999

[25] Campbell A., De Meer H., Kounavis M., Miki K., Vicente J., Villela D., *A Survey of Programmable Networks*, SIGCOMM Computer Communication Review Vol. 29, 04/1999

[26] Campione M., Walrath K., *The Java tutorial: Second Edition*, Sun Microsystem Press, Mountain view, USA, 1998

[27] Casner S., Frederick R., Jacobson V., Schulzrinne H., *RTP: A Transport Protocol for Real-Time Applications*, RFC1889, 01/1996

[28] Chandy K., Kiniry J., Rifkin A., Zimmermann D., Tanaka W., Weisman L., *A Framework for Structured Distributed Object Computing*, Parallel Computing, Vol. 24, No. 12-13, 1901-1922, 11/1998

[29] Charbert A., Grossman E., Jackson L., Pietrowicz S., Seguin C., *Java Object-Sharing in Habanero*, ACM, Communications of, Vol. 41, No. 6, 06/1998

[30] Dommel H., Garcia-Luna-Aceves J., *Group Coordination Support for Synchronous Internet Collaboration*, IEEE, Internet Computing, 03+04/1999

[31] Dunstan S., Morrison G., Nilsson M., Okubo S., Radha H., Skran D., Thom G., *ITU-T Standardization of Audiovisual Communication Systems in ATM and LAN Environments*, IEEE, Journal on Selected Aereas in Communications, Vol. 15, No. 6, 08/1997

[32] Eggers O., *Entwurf und Realisierung einer Konferenzsteuerungseinheit mit einer dynamisch adaptiven Benutzeroberfläche*, Studienarbeit, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 04/1999

[33] Eggers O., Sturzebecher D., Zitterbart M., *IBR-Whiteboard: An advanced whiteboard to improve tele-teaching*, IN-TELE, Strasbourg, France, 09/1998

[34] Eggers O., *Transaktionsbasierte Sitzungskontrolle*, Diplomarbeit, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 07/2000

[35] Ellis C., Gibbs S., Rein G., *Groupware - Some Issues and Experience*, Communications of the ACM, Vol. 34, No. 1, 1991

[36] Geary D., *Graphic Java - mastering the AWT*, Sun Microsystem Press, Mountain view, USA, 1997

[37] Georganas N., Oliveira J., Shirmohammadi S., "*Applet-Based Telecollaboration: A Network-centric Approach*", IEEE Multimedia Magazine, Volume 5, Number 2, 04-06/1998

[38] Georganas N., Shirmohammadi S., *JETS: a Java-Enabled Telecollaboration System*, IEEE, ICMCS, Los Alamitos, USA, 1997

[39] Gettys J., Scheifler R., *X Windows System*, digital Press/Prentice Hall, USA, 1992

[40] Geyer W., *Das digital lecture board*, Band 58, Infix-Verlag, St. Augustin, 1999

[41] Greenberg S., *Computer-Supported Cooperative Work and Groupware: An Introduction into Special Issues*, Int. Journal on Machine Studies, Vol.34, No.2, 1991

[42] Grote H., *Einsatz von Servergruppen zur Erweiterung der Skalierbarkeit von Kollaborationssystemen*, Diplomarbeit, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 09/2000

[43] Gutfreund Y., Martin C., Nicol J., Paschetto J., Rush K., *Collaborative Multimedia Applications*, ACM, Communications of, Vol. 42, No. 1, 01/1999

[44] Handley M., *Internet Draft - SAP: Session Announcement Protocol*, IETF, MMUSIC, 11/1996

[45] Handley M., Jacobson V., *Internet Draft - SDP: Session Description Protocol*, IETF, MMUSIC, 04/1998

[46] Handley M., Rosenberg J., Schooler E., Schulzrinne H., *SIP: Session Initiation Protocol*, RFC 2543, 03/1999

[47] Hehman J., *Linux in a Nutshell*, O'Reilly, Sebastopol, USA, 1997

[48] Helbig T., Trossen D., *Die ITU Standrad-Familie T.120 als Basis für verteilte Mehrbenutzersysteme*, KiVS, Braunschweig, Germany, 02/1997

[49] Hofmann M., *Skalierbare Multicast-Kommunikation in Weitverkehrsnetzen*, infix, St. Augustin, 1998

[50] Howes T., Kille S., Wahl M., *Lightweight Directory Access Protocol (v3)*, RFC 2251, 12/1997

[51] Hutchison C., Rosenberg D., *Desing Issues in CSCW*, Springer Verlag, London, UK, 1994

[52] ITU-T, *Recomendation H.225.0: Call signalling protocols and media stream packetization for packet-based multimedia communication systems*, ITU, 02/1998

[53] ITU-T, *Recomendation H.245: Control protocol for multimedia communication*, ITU, 09/1998

[54] ITU-T, *Recomendation H.323:Packet-based multimedia communication systems*, ITU, 02/1998

[55] ITU-T, *Recomendation T.120: Data protocols for multimedia conferencing*, ITU, 07/1996

[56] ITU-T, *Recomendation T.121: Generic application template*, ITU, 07/1996

[57] ITU-T, *Recomendation T.122: Multipoint Communication Services*, ITU, 02/1998

[58] ITU-T, *Recomendation T.124: Generic Conference Control*, ITU, 02/1998

[59] ITU-T, *Recomendation T.125: Multipoint Communication Services*, ITU, 02/1998

[60] ITU-T, *Recomendation T.126: Multipoint still image and annotation protocol*, ITU, 07/1997

[61] ITU-T, *Recomendation T.127: Multipoint binary file transfer*, ITU, 08/1995

[62] ITU-T, *Recomendation X.500: Open System Ineterconnection -The Directory: Overview of concepts, models, and services*, ITU, 1993

[63] Kamoun F., Saidane L., *Performance Evaluation and Management of the Consensus Algorithm in a Realtime Distributed Transactional System*, Computer Communications, Red Sea, Egypt, 07/1999

[64] Kuck M., *Integration von LDAP in MACS*, Diplomarbeit, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 06/1999

[65] Kumar V., *MBone - Interactive Multimedia on the Internet*, New Riders, Indianapolis, USA, 1996

[66] Lea D., *Concurrent Programming in Java - Design Principals and Patterns*, Addison-Wesley Longman, 1997

[67] MACS Team, *The MACS API description*, http://macs.ibr.cs.tu-bs.de/, 2000

[68] Mahalek W., *Telekollaboration in der Software Entwicklung*, doctoral thesis, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, in preparation

[69] McCanne S., *Scaleable Multimedia Communication*, IEEE, Internet Computing, 03+04/1999

[70] Messerschmitt D., *Networked Applications*, Morgan Kaufmann, San Fransisco, USA, 1999

[71] Messerschmitt D., *The Prospect for Computing Communication Convergence*, VISION 21, Munich, Germany, 11/1999

[72] Messerschmitt D., *Understanding Networked Applications*, Morgan Kaufmann, San Francisco, USA, 2000

[73] Miao Y., Pfister H., Wessner M., *Using learning Protocols to Structure Computer-Supported Cooperative Learning*, ED-MEDIA, Seattle, USA, 06/1999

[74] Miller C., *Multicast Networking and Applications*, Addison-Wesley, Massachusetts, USA, 1998

[75] Minden G., Sincoskie W., Smith J., Tennenhouse D., Wetherall D., *A Survey of Active Network Research*, ICM Vol. 35 No. 1, 1997

[76] Nahrstedt K., Steinmetz R., *Multimedia: Computing, Communications and Applications*, Prentice Hall, Upper Saddle River, USA, 1995

[77] Oaks S., *Java Security*, O'Reilly, Sebastopol, USA, 1998

[78] Oldiges C., *Anbindung ITU-basierter Konferenzsysteme*, Diplomarbeit, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 02/2000

[79] Ott J., Perkins C., Kutscher D., *Internet Draft - A Message Bus for Conferencing Systems*, IETF, MMUSIC, 08/1998

[80] Ott J., Perkins C., Kutscher D., *Internet Draft - Requirements for Local Conference Control*, IETF, MMUSIC, 12/1999

[81] Parnes P., Synnes K., Schefström D., *The CDT mStar environment: Scalable Distributed Teamwork in action*, Group, 1997, Phoenix, USA, 11/1997

[82] Pendergast M., *A Comparative Analysis of Groupware Application Protocols*, ACM, Computer Communication Review, Vol. 28, No. 1, 01/98

[83] Pupolin S., Natale F., *Multimedia Communications*, Springer, London, UK, 1999

[84] Rapaport M., *Computer Mediated Communication*, Wiley, New York, USA 1991

[85] Richardson J., Riley M., *Digital Video Communication*, Artech House, Norwood, USA, 1997

[86] Rosenberg J., Schulzrinne H., *A compariosn of SIP and H323 for Internet Telephony*, International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Cambridge, England, July 1998

[87] Schaphorst R., *Videoconferencing and Videotelephony*, Artech House, Norwood, USA, 1999

[88] Schulzrinne H., *Dynamic Configuration of Conference Applications using Pattern-Matching Multicast*, NOSSDAV, Durham, England, 04/1995

[89] Seemann J., Wolff von Gudenberg J., *Software-Entwurf mit UML*, Springer Verlag, Berlin, 2000

[90] Stallings W., *Network and Internetwork Security*, New Jersy, USA, 1995

[91] Steinmetz R., *Multimedia - Technologie*, Springer, Berlin, Germany, 1993

[92] Sturzebecher D., *MACS technical report*, Institute of Operating Systems and Computer Networks, Technical University of Braunschweig, Germany, 06/2000

[93] Sun Microsystems, *Java HotSpot Technology*, http://java.sun.com/hotspot, 1999

[94] Sun Microsystems, *Java ™ Media Framework API guide*, Sun Microsystems, California, USA, 11/1999, available from http://java.sun.com/jmf

[95] T. Liao. *LRMP: the light-weight reliable multicast protocol*, http://monet.inria.fr/, 1997

[96] Thom G., *H.323: The Multimedia Communications Standard for Local Area Networks*, IEEE, Communications Magazine, 12/1996

[97] Thomas P., *CSCW Requirements and Evaluation*, Springer Verlag, London, UK, 1996

[98] W. Crawford, D. Flanagan, J. Farley, K. Magnusson, *Java Enterprise Edition in a Nutshell*, O'Reilly, Sebastopol, USA, 1999

[99] Willers M., *Implementierung von Floorkontroll-Strategien für das Kollaborationssystem MACS*, Studienarbeit, Institute of Operating Systems and Computer Networks, TU-Braunschweig, Germany, 08/1999

[100] Wittmann R., Zitterbart M., *Active multicasting for heterogeneous groups*, 4th International Conference on Broadband Communications, Stuttgart, Germany, 04/1998

[101] Wittmann R., Zitterbart M., *Multicast Communication: Protocols, Programming and Applications*, Morgan Kaufman Publischers, San Fransisco, USA, 2000